PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-284995
(43)Date of publication of application : 13.10.2000

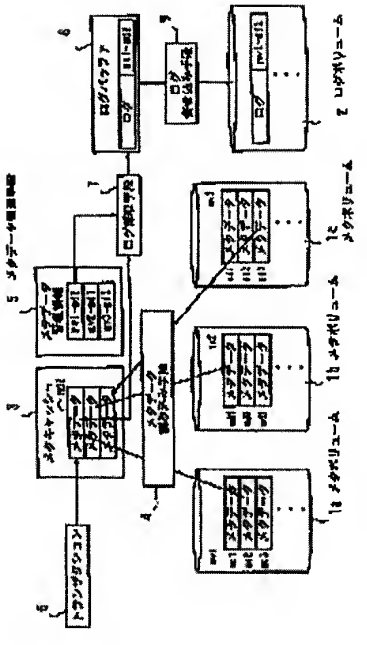
(51)Int.Cl. G06F 12/00

(21)Application number : 11-087457
(22)Date of filing : 30.03.1999

(71)Applicant : FUJITSU LTD
(72)Inventor : KOSEKI MICHIHIKO
YOKOYAMA MAMORU
WASHIMI MASASHI
YAMAGUCHI SATORU
TANIWAKI SADAYOSHI
HAMANAKA SEISHIRO

(54) DATA PROCESSOR AND RECORDING MEDIUM

(57)Abstract:
PROBLEM TO BE SOLVED: To attain efficient file system restoration processing.
SOLUTION: When meta data are updated by a transaction 6, necessary meta data in meta volumes 1a-1c are read in a cache 3 by a meta data reading means 4. Which volume the read meta data are read from is managed by a meta data managing means 5. In this case, when the contents of the meta data are updated by the transaction 6, the updated meta data are adopted as a log by a log collecting means 7. In this case, the identification information of the meta volumes 1a-1c in which the meta data are stored is collected. The collected information is held in a log buffer 8. Then, the information in the log buffer 8 is written in a log volume 2 by a log writing means 9.



(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（A）**　(11)特許出願公開番号
特開2000－284995
（P2000－284995A）
(43)公開日　平成12年10月13日(2000.10.13)

(51)Int.Cl.[7]	識別記号	ＦＩ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　12/00	５１４ ５３１	Ｇ０６Ｆ　12/00	５１４Ａ　５Ｂ０８２ ５３１Ｊ

審査請求　未請求　請求項の数21　ＯＬ　（全 81 頁）

(21)出願番号　特願平11－87457

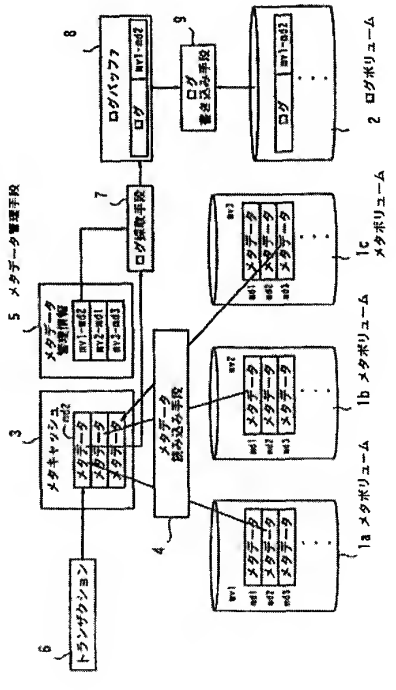
(22)出願日　平成11年３月30日(1999.3.30)

(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号
(72)発明者　小関　道彦
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(72)発明者　横山　衛
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(74)代理人　100092152
弁理士　服部　毅巖

最終頁に続く

(54)【発明の名称】　データ処理装置及び記録媒体

(57)【要約】
【課題】　ファイルシステム修復処理の効率化を図る。
【解決手段】　トランザクション６がメタデータの更新をする際には、まず、メタデータ読み込み手段４によりメタボリューム１ａ～１ｃ内の必要なメタデータがメタキャッシュ３に読み込まれる。その際、読み込んだメタデータがどのメタボリュームから読み込まれたものであるのかが、メタデータ管理手段５によって管理される。ここで、トランザクション６がメタデータの内容を更新すると、ログ採取手段７によって更新後のメタデータがログとして採取される。この際、メタデータが格納されていたメタボリューム１ａ～１ｃの識別情報をも採取する。採取した情報は、ログバッファ８に保持される。そして、ログ書き込み手段９により、ログバッファ８内の情報がログボリューム２に書き込まれる。



1

【特許請求の範囲】

【請求項１】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、

ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリュームと、

メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、

メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、

メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、

前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、

前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段と、

前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段と、

前記ログ採取手段が採取した情報を保持するログバッファと、

前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、

を有することを特徴とするデータ処理装置。

【請求項２】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、

ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、

メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、

メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、

メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、

前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、

前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、

前記ログ採取手段が採取した情報を保持するログバッファと、

前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、

前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、

前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段と、

ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、

を有することを特徴とするデータ処理装置。

2

【請求項３】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、

ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、

メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、

メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、

メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、

前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、

前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、

前記ログ採取手段が採取した情報を保持するログバッファと、

前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、

前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、

前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段と、

前記ログ書き込み手段がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶手段に格納されたシーケンス番号を基準として採番するシーケンス番号採番手段と、

を有することを特徴とするデータ処理装置。

【請求項４】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、

ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、

メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、

メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み

込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段と、
を有することを特徴とするデータ処理装置。

【請求項５】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段と、
前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作用管理情報とするとともに、前記獲得操作用管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作用管理情報とする獲得操作用管理情報生成手段と、
メタデータの獲得及び解放要求を出力するトランザクションと、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作用管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作用管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段と、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、
を有することを特徴とするデータ処理装置。

【請求項６】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段と、
メタデータの獲得及び解放要求を出力するトランザクションと、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記割り当て管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段と、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、
前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段と、
を有することを特徴とするデータ処理装置。

【請求項７】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクションと、
ログをトランザクション毎に保持する、サイズの異なる複数のログバッファと、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段と、
を有することを特徴とするデータ処理装置。

【請求項８】　前記ログ採取手段は、最初にログを格納する場合には、トランザクションの内容によって予想される処理に適した大きさの前記ログバッファに格納し、格納対象となる前記ログバッファの記憶容量が不足してきたら、より大きな記憶容量のログバッファへログを移し替え、以後、より大きな記憶容量のログバッファをログの格納対象とすることを特徴とする請求項７記載のデータ処理装置。

【請求項９】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
ログをトランザクション毎に保持するログバッファと、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段と、
前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段と、
を有することを特徴とするデータ処理装置。

【請求項１０】　前記ログ書き込み手段は、前記中間ロ

グに対して、トランザクションを実行するのに必要とされたパラメタに関する情報を付加し、
ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログを用いて、前記メタボリューム内のメタデータの不整合を修正するとともに、前記中間ログを発見すると、前記中間ログに含まれたパラメタを用いて、前記トランザクションを再実行させるファイルシステム復元手段をさらに有することを特徴とする請求項９記載のデータ処理装置。
【請求項１１】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクションと、
前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ許否を判断するトランザクション受け入れ判断手段と、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、
前記ログ採取手段が採取した情報を保持するログバッファと、
前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、
を有することを特徴とするデータ処理装置。
【請求項１２】　前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、
前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段とをさらに有し、
前記トランザクション受け入れ判断手段は、前記有効範囲監視手段によって有効なログとされたログがログボリューム中に占める割合が一定値以上である間は、前記トランザクションの受け入れを拒絶することを特徴とする請求項１１記載のデータ処理装置。
【請求項１３】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段、
前記ログ採取手段が採取した情報を保持するログバッファ、
前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。
【請求項１４】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段、
前記ログ採取手段が採取した情報を保持するログバッファ、
前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段、
前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段、
前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のロ

グを、有効なログとして指定する有効範囲監視手段、
ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１５】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段、
前記ログ採取手段が採取した情報を保持するログバッファ、
前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段、
前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段、
前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段、
前記ログ書き込み手段がログの書き込みを行う際に、システムの使用可能年数以上使い続けることができる値を最大値としたシーケンス番号を昇順で採番し、書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶手段に格納されたシーケンス番号から昇順で採番するシーケンス番号採番手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１６】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１７】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段、
前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作用管理情報とするとともに、前記獲得操作用管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作用管理情報とする獲得操作用管理情報生成手段、
メタデータの獲得及び解放要求を出力するトランザクション、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作用管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作用管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１８】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段、
メタデータの獲得及び解放要求を出力するトランザクション、
前記トランザクションによりメタデータの獲得要求が出

された場合には、前記割り当て管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段、
前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。
【請求項１９】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクション、
ログをトランザクション毎に保持する、サイズの異なる複数のログバッファ、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。
【請求項２０】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
ログをトランザクション毎に保持するログバッファ、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段、
前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。
【請求項２１】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクション、
前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ許否を判断するトランザクション受け入れ判断手段、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段、
前記ログ採取手段が採取した情報を保持するログバッファ、
前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明はファイルシステムの修正機能を有するデータ処理装置及び記録媒体に関し、特にログを用いてファイルシステムの不整合の修正を行うデータ処理装置及び記録媒体に関する。
【０００２】
【従来の技術】コンピュータシステムを運用していると、何らかの理由によりシステムがダウンする場合がある。突然のシステムダウンが発生すると、ファイルシステムの不整合が生じる。そこで、システムダウン後のブート時には、従来であればファイルシステム全体を走査

し、その矛盾点の検出を行う。矛盾点が発見されたら、場合に応じた変更をファイルシステムに加えることによって、ファイルシステムの整合性を回復していた。ところが、ファイルシステム全体を走査するには多くの時間が必要であり、結果としてシステムダウン後の復旧の遅れを招いていた。

【０００３】そこで近代のＵＮＩＸオペレーティングシステム（ＯＳ）のような多くのコンピュータＯＳのファイルシステムでは、ファイルシステムオペレーションにおいてファイルシステムに保存されているデータが更新される度に、その更新情報をログ（ジャーナル）として採取している。ファイルシステムオペレーション時に更新情報のログ採取を行っておくことにより、システムダウン後のブート時のファイルシステム整合性チェックのフェーズでは、残されたログを順次走査し、対応する領域へアップデートすることによって、ファイルシステムの整合性が保証される。その結果、システムのダウン時間が短縮される。

【０００４】

【発明が解決しようとする課題】しかし、ログ機構を導入することにより、次に挙げる問題が生じていた。

（１）第１の問題点

ファイルを管理するメタデータ（二次記憶装置に格納されたファイルを管理するためのデータ）は、ファイルシステムオペレーション時に二次記憶装置からメモリへ読み込まれ、メモリ内において操作される。その後、所定のタイミングで二次記憶装置へとその更新内容が反映される。ログ機構の導入時には、その二次記憶装置への反映前に更新内容をログ専用の二次記憶装置へと記録する必要がある。

【０００５】しかし、大規模ファイルシステムに対応するために、複数の二次記憶装置がメタデータに割り当てられ、異なる二次記憶装置に割り当てられたメタデータを１つのファイルシステムオペレーションが操作する場合がある。このファイルシステムオペレーションのログを採取する際に、メモリ内のメタデータだけをログとして記録していたのでは、残されたログからメタデータ毎に異なる二次記憶装置を検索するのに時間を消費し、ログ機構導入による最大のメリットである、ファイルシステム復元の時間短縮に悪い影響を与える。

【０００６】（２）第２の問題点

同様にファイルシステム復元の時間短縮に悪い影響を与える要因として、有限なサイズしかないログ専用の二次記憶装置全体をファイルシステム復元時に探索することが考えられる。

【０００７】ログ機構の導入はファイルシステムオペレーションを細分化したトランザクションの完了を常に保証しつつ動作するため、ログ機構を導入した多くのファイルシステムはトランザクションにシーケンス番号を与え、ファイルシステム復元時にはシーケンス番号をもとに最古のトランザクションを同定する。そして、最古のトランザクションからログリプレイと呼ばれるログを用いたファイルシステム復元操作を開始する。

【０００８】ここで、有限サイズのログ専用の二次記憶装置内には、ファイルシステムの整合性を復元するために欠かせないログが確かに存在する可能性があるが、有限なサイズを有効利用するためにログ専用の二次記憶装置は過去のログを上書きしてサイクリックに利用しなければならない。このような処理を行うには、ある程度以上古いログが必ず不必要となっていることが前提となる。従って、多くのログの内容は既に不必要となっている。すなわち、ログ専用の二次記憶装置全体を探索あるいは反映するのは非効率的である。

【０００９】（３）第３の問題点

最古のトランザクションを特定するためのシーケンス番号は単調増加であることが要求され、運用途中にオーバーフローによってゼロに戻されてしまうことは許されない。これを回避するために、ファイルシステム復元作業終了時、またはオーバーフロー直前にログ専用の二次記憶装置全体をゼロクリアし、再度シーケンス番号ゼロから順にトランザクションを処理するのが一般的である。しかし、ログ専用の二次記憶装置をゼロクリアするためには多くの時間が必要となる。少なくともシステムの運用を一時停止する必要があり、サーバ装置などによるサービスの提供を停止せざるを得なくなってしまう。

【００１０】以上の（１）～（３）の課題はファイルシステム復元時の問題であるが、ログ機構を導入することは通常の運用時にも問題を引き起こす可能性を持っている。ログ機構は、メモリ上にログをスプールする手法や、ログ専用の二次記憶装置として用いられるディスクの特性を考慮したシーケンシャルアクセスなど、高速化のための条件は整えられているが、ログの採取の仕方を熟考しなければ、ログ採取に伴う性能劣化は非常に大きいものとなりうる。

【００１１】（４）第４の問題点

単一のトランザクション内で、同一データを複数回更新することは度々あるが、その都度、その同一データに対するログを採取したのではメモリが不足し、二次記憶装置へのＩ／Ｏ量が増加する。

【００１２】（５）第５の問題点

ログ機構の導入はトランザクションの順序性を保証し、終了したトランザクションのログを順に採取することを要求するために、あるトランザクションが操作したログ対象データを他のトランザクションが操作することが一般的に不可能な状況となりうる。ここで、個々のファイルの内容を対象とするトランザクションについては、ファイル単位に排他制御を行うことによって、複数のトランザクションが並列実行することは比較的容易である。しかしながら、個々のファイルによらないもの、特に領域の割り当て情報を操作する場合には、並列実行がきわ

めて難しくなる。

【００１３】領域の獲得・解放処理が並列に動作する場合を考えると、それぞれが獲得、解放のログを採取する。同一の管理情報（ここでは、ビットマップを考える）によって管理される領域の獲得・解放処理であった場合には、同一の管理情報が、解放された状態、獲得された状態でログに記録されることになる。

【００１４】ファイルシステムの復元処理を行わなければならないようなシステムダウンが発生するタイミングによっては、このように別トランザクションの更新情報を含むログの採取方式では様々な問題が生じる。

【００１５】Ａというトランザクションが解放処理、Ｂというトランザクションが獲得処理を行う場合を考える。ここで、トランザクションＡの解放処理はトランザクションＢの獲得処理よりも先に行われ、かつ、トランザクションＡはトランザクションＢよりも後に終わる場合を例に挙げる。

【００１６】このとき、トランザクションＡが開放した領域をトランザクションＢが獲得してしまう場合が考えられる。トランザクションＢが先に終了することから、残されたログには、まず獲得処理が記録され、次に解放処理が記録される。

【００１７】上記の場合のトランザクションＢのログだけが記録されており、それを復元に用いた場合には、本来行われたはずである解放処理の記録が残されていないことから、該当領域が二重に獲得された状態となってしまう。トランザクションＡのログまで記録されており、復元に用いられると、トランザクションＢが利用している領域がトランザクションＡの解放処理のログによって解放されている状態となってしまう。いずれも本来の状態とは異なっており、避けなければならない。しかしながら、これを回避するために並列実行を制限することは、マルチタスクを実現したＯＳ上のファイルシステムの速度性能の低下に与える影響が非常に大きい。

【００１８】（６）第６の問題点

既に述べたようにログ機構の第一の目的としてファイルシステムを復元するために費やす時間の短縮が挙げられるが、そのためにトランザクションの独立性を疎かにしたり、中途半端な状態での整合性回復によりあたかも正常に動作しているかのように振る舞うファイルシステムが多く見受けられる。

【００１９】従来のメタデータ管理方式のように、ログを記録するためのメモリ空間をファイルシステム全体で１つのキャッシュメモリを共有していたのではトランザクション毎の独立性を保つのが難しく、他のトランザクションによる更新情報が１つのトランザクションのログとして記録されてしまう可能性が高い。特にファイルに対する排他で制御しきれない割り当て管理情報については前述の通りである。

【００２０】（７）第７の問題点

トランザクション毎に必要とするログのサイズが異なることから、複数のログバッファとしてログを記録するためのメモリをトランザクション毎に割り当て、管理する場合に、全てのメモリサイズが同一である必要はない。例えば、ファイルの参照時刻を更新するだけのトランザクションが残すログのサイズは大変小さく、巨大なデータ書き込み要求のトランザクションは必然的にそれだけ大きいログサイズとなる。

【００２１】（８）第８の問題点

トランザクション毎に残すログサイズの違いを考慮して、限られたメモリ空間の有効利用を試みても、キャッシュメモリサイズは残されるログサイズに比較して、やはり小さい。

【００２２】（９）第９の問題点

「第８の問題点」の解決策として、１つのトランザクションを分割し、中途の状態のログを出力することが考えられる。しかし、一般にファイルを管理するメタデータは、トランザクションが中途の状態ではやはり中途の状態であり、そのままログに記録したところで、そのログを用いて復元されるファイルシステムは中途の状態にしかなり得ない。これではオペレーションのセマンティクスを保証した復元とはなり得ない。

【００２３】（１０）第１０の問題点

「第９の問題点」で示した中途の状態でのトランザクションの中断はファイルシステムにとって危険な状態といえる。ログ機構の導入は、採取したログをログボリュームに反映するまでは該当メタデータもキャッシュメモリから追い出せないことを意味する。ログがキャッシュメモリに入りきらないほど巨大となっているときには、メタデータを管理するキャッシュメモリの利用率も高くなっていることが考えられる。ログを出力するまでメタデータをキャッシュメモリから追い出せないため、このような状態で多くのトランザクションが並列実行されると、メモリ枯渇によるハングアップ状態に陥ることも考えられる。

【００２４】（１１）第１１の問題点

第１０の問題点と同様の資源枯渇はログ記録用の二次記憶装置についても言える。キャッシュメモリに比較すれば巨大な二次記憶装置についても、メタデータ記録用の二次記憶装置へのＩ／Ｏを削減するために、多くのログを有効なログとして残しておけば、それは利用可能な領域の減少を引き起こす。その際に多数のトランザクションの並列実行を許すことはメタデータキャッシュ枯渇、ログキャッシュ枯渇、ログ記録用二次記憶装置枯渇、などを誘発する可能性がある。

【００２５】本発明はこのような点に鑑みてなされたものであり、ファイルシステム修復処理の効率化を図ったデータ処理装置を提供することを目的とする。

【００２６】

【課題を解決するための手段】本発明では上記課題を解

決するための第１の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段と、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【００２７】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。その際、読み込んだメタデータがどのメタボリュームから読み込まれたものであるのかが、メタデータ管理手段によって管理される。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。この際、メタデータが格納されていたメタボリュームの識別情報をも採取する。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。

【００２８】また、上記課題を解決する第２の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段と、ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、を有することを特徴とするデータ処理装置が提供される。

【００２９】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。その後、メタデータ書き込み手段により、メタキャッシュ内のメタデータがメタボリューム内に格納される。その書き込み動作は、有効範囲監視手段で監視されており、変更内容がメタボリュームに反映されていないメタデータに対応するログボリューム内のログが、有効なログとして指定される。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段により、ログボリュームに格納されたログの中で、有効範囲監視手段により有効なログとして指定されているログのみを用いて、メタボリューム内のメタデータの不整合が修正される。

【００３０】また、上記課題を解決する第３の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、前記ファイルシステム復元手段が前記メタボリューム内のメタデータ

の不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段と、前記ログ書き込み手段がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶手段に格納されたシーケンス番号を基準として採番するシーケンス番号採番手段と、を有することを特徴とするデータ処理装置が提供される。

【００３１】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。その際、シーケンス番号採番手段により、シーケンス番号が昇順で採番され、書き込むべきログに付与される。ファイルシステムに不整合が発生すると、ファイルシステム復元手段により、ログボリュームに残されたログを用いてメタデータの不整合が修正される。このとき、修正に用いられた最後のログのシーケンス番号が初期シーケンス番号記憶手段に記憶される。その後、ログ書き込み手段によりログバッファ内の情報がログボリュームに書き込まれると、シーケンス番号採番手段により、初期シーケンス番号記憶手段に格納されたシーケンス番号を基準としてシーケンス番号が採番され、書き込むべきログに付与される。

【００３２】また、上記課題を解決する第４の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【００３３】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新する。すると、ログ採取手段により、トランザクションの種別が判断され、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、メタキャッシュ内で変更されたメタデータの最終形態のみがログとして採取される。

【００３４】また、上記課題を解決する第５の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段と、前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作用管理情報とするとともに、前記獲得操作用管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作用管理情報とする獲得操作用管理情報生成手段と、メタデータの獲得及び解放要求を出力するトランザクションと、前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作用管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作用管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段と、前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、を有することを特徴とするデータ処理装置が提供される。

【００３５】このようなデータ処理装置によれば、獲得操作用管理情報生成手段により、割り当て管理情報の一部領域の複製が生成され、獲得操作用管理情報とされる。トランザクションにより獲得要求があると、メタデータ獲得手段により、獲得操作用管理情報内の未獲得のメタデータが獲得される。すると、獲得操作用管理情報と割り当て管理情報との内容が更新される。また、トランザクションよりメタデータの解放要求があると、メタデータ解放手段により該当するメタデータの解放処理が行われる。この際、割り当て管理情報の内容のみが更新される。ここで、獲得操作用管理情報内に未獲得のメタデータがなくなると、割り当て管理情報内の別の領域の複製が獲得操作用管理情報とされる。

【００３６】また、上記課題を解決する第６の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段と、メタデータの獲得及び解放要求を出力するトランザクションと、前記トランザクションによりメタデータの獲得要求が出された場合には、前記割り当て管理情報保の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段

と、前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【００３７】このようなデータ処理装置によれば、トランザクションからメタデータの獲得要求が出されると、メタデータ獲得手段によって未獲得のメタデータの１つが割り当て管理情報内から獲得され、そのメタデータが獲得済みの状態とされる。また、トランザクションからメタデータの解放要求が出されると、メタデータ解放手段によって該当するメタデータが未獲得の状態に変更される。そして、ログ採取手段により、割り当て管理情報内のメタデータ獲得手段及びメタデータ解放手段によって変更された部分の情報がログとして採取される。

【００３８】また、上記課題を解決する第７の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクションと、　ログをトランザクション毎に保持する、サイズの異なる複数のログバッファと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【００３９】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新する。すると、ログ採取手段により、メタキャッシュ内で変更されたメタデータがログとして採取され、トランザクション毎のログバッファに保持される。

【００４０】また、上記課題を解決する第８の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、ログをトランザクション毎に保持するログバッファと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段と、前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【００４１】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログバッファに格納しきれなくなるか、トランザクションが終了すると、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。ログバッファに格納しきれなくなった場合には、ログバッファの内容を完結したログに加工され、中間ログとしてログボリュームに格納される。

【００４２】また、上記課題を解決する第９の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクションと、前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ許否を判断するトランザクション受け入れ判断手段と、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【００４３】このようなデータ処理装置によれば、トランザクションから開始要求が出されると、トランザクション受け入れ制限手段が受け入れの許否を判断する。その後、メタデータ書き込み手段によるメタデータの書き

込みが進み、有効なログの割合が減少したら、その時点でトランザクションの開始要求を許可する。

【００４４】

【発明の実施の形態】以下、本発明の実施の形態を図面を参照して説明する。図１は、第１の発明の原理構成図である。この第１の発明は、複数の二次記憶装置がメタデータに割り当てられている場合におけるファイルシステム不整合修復時間の短縮を図るものである。

【００４５】このデータ処理装置には、二次記憶装置としてメタボリューム１ａ，１ｂ，１ｃとログボリューム２とが設けられている。各メタボリューム１ａ，１ｂ，１ｃには、ファイルを管理するためのメタデータが記憶されている。また、ログボリューム２には、メタデータの更新結果であるログが記憶されている。また、主記憶装置内にはメタキャッシュ３が設けられている。メタキャッシュ３は、メタデータを記憶するための主記憶装置内の記憶領域である。メタデータ読み込み手段４は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ３へと読み込む。メタデータ管理手段５は、メタキャッシュ３に読み込まれたメタデータが格納されていたメタボリューム１ａ，１ｂ，１ｃの識別情報を管理する。トランザクション６は、メタキャッシュ３内に読み込まれたメタデータの内容を更新する。ログ採取手段７は、メタキャッシュ３内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリューム１ａ，１ｂ，１ｃの識別情報を採取する。ログバッファ８は、ログ採取手段７が採取した情報を保持する。ログ書き込み手段９は、ログバッファ８が保持する情報を、適宜ログボリューム２に格納する。

【００４６】このようなデータ処理装置よれば、トランザクション６がメタデータの更新をする際には、まず、メタデータ読み込み手段４によりメタボリューム１ａ～１ｃ内の必要なメタデータがメタキャッシュ３に読み込まれる。その際、読み込んだメタデータがどのメタボリュームから読み込まれたものであるのかが、メタデータ管理手段５によって管理される。ここで、トランザクション６がメタデータの内容を更新すると、ログ採取手段７によって更新後のメタデータがログとして採取される。この際、メタデータが格納されていたメタボリューム１ａ～１ｃの識別情報をも採取する。採取した情報は、ログバッファ８に保持される。そして、ログ書き込み手段９により、ログバッファ８内の情報がログボリューム２に書き込まれる。

【００４７】これにより、ログボリューム２に保持されたログがどのメタボリューム１ａ～１ｃのメタデータに関するログであるのかを管理することができる。その結果、複数のメタボリューム１ａ～１ｃにメタデータが格納されていても、ファイルシステムの不整合を修正することが可能となる。

【００４８】図２は、第２の発明の原理構成図である。第２の発明は、ログの有効範囲を監視することで、ファイルシステム復元時の効率化を図ったものである。第２の発明は、以下のような要素で構成される。

【００４９】メタボリューム１１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム１２は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ１３は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段１４は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ１３へと読み込む。トランザクション１５は、メタキャッシュ１３内に読み込まれたメタデータの内容を更新する。ログ採取手段１６は、メタキャッシュ１３内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取する。ログバッファ１７は、ログ採取手段１６が採取した情報を保持する。ログ書き込み手段１８は、ログボリューム１２内の領域を定期的に循環するようにして、ログバッファ１７が保持する情報をログボリューム１２に格納する。メタデータ書き込み手段１９は、メタキャッシュ１３内のメタデータをメタボリューム１１内に格納する。有効範囲監視手段２０は、メタデータ書き込み手段１９による書き込み動作を監視しており、変更内容がメタボリューム１１に反映されていないメタデータに対応するログボリューム１２内のログを、有効なログとして指定する。ファイルシステム復元手段２１は、ファイルシステム復元要求を受け取ると、ログボリューム１２に格納されたログの中で、有効範囲監視手段２０により有効なログとして指定されているログのみを用いて、メタボリューム１１内のメタデータの不整合を修正する。なお、有効範囲監視手段２０による有効なログの指定方法としては、例えば、有効範囲を示す情報を不揮発性の記録媒体（ログボリューム１２等）に記録することができる。この場合、ファイルシステム復元手段２１は、有効範囲が記録された不揮発性の記録媒体内の情報を読みとることで、有効なログと指定されているログを認識できる。

【００５０】このようなデータ処理装置によれば、トランザクション１５がメタデータの更新をする際には、まず、メタデータ読み込み手段１４によりメタボリューム１１内の必要なメタデータがメタキャッシュ１３に読み込まれる。ここで、トランザクション１５がメタデータの内容を更新すると、ログ採取手段１６によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ１７に保持される。そして、ログ書き込み手段１８により、ログバッファ１７内の情報がログボリューム１２に書き込まれる。その後、メタデータ書き込み手段１９により、メタキャッシュ１３内のメタデータがメタボリューム１１内に格納される。その書き込

み動作は、有効範囲監視手段２０で監視されており、変更内容がメタボリューム１１に反映されていないメタデータに対応するログボリューム１２内のログが、有効なログとして指定される。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段２１により、ログボリューム１２に格納されたログの中で、有効範囲監視手段２０により有効なログとして指定されているログのみを用いて、メタボリューム１１内のメタデータの不整合が修正される。

【００５１】これにより、ファイルシステムを復元する際には、ログボリューム１２内の有効なログのみを用いて効率よく復元処理を行うことが可能となる。図３は、第３の発明の原理構成図である。第３の発明は、ログに付加するシーケンス番号のデータサイズを十分大きなものとし、シーケンス番号を常に昇順で使い続けることができる（ゼロクリアが不要となる）ようにしたものである。第３の実施の形態は、以下のような要素で構成される。

【００５２】メタボリューム３１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム３２は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ３３は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段３４は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ３３へと読み込む。トランザクション３５は、メタキャッシュ３３内に読み込まれたメタデータの内容を更新する。ログ採取手段３６は、メタキャッシュ３３内で変更されたメタデータの内容をログとして採取する。ログバッファ３７は、ログ採取手段３６が採取した情報を保持する。ログ書き込み手段３８は、ログバッファが保持する情報を前記ログボリュームに格納する。ファイルシステム復元手段３９は、ログボリューム３２に格納されたログを用いてメタボリューム３１内のメタデータの不整合を修正する。初期シーケンス番号記憶手段３０ａは、ファイルシステム復元手段３９がメタボリューム３１内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する。シーケンス番号採番手段３０ｂは、ログ書き込み手段３８がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、ファイルシステム復元手段３９がメタボリューム３１内のメタデータの不整合を修正した直後には、初期シーケンス番号記憶手段３０ａに格納されたシーケンス番号を基準として採番する。

【００５３】このようなデータ処理装置によれば、トランザクション３５がメタデータの更新をする際には、まず、メタデータ読み込み手段３４によりメタボリューム３１内の必要なメタデータがメタキャッシュ３３に読み込まれる。ここで、トランザクション３５がメタデータの内容を更新すると、ログ採取手段３６によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ３７に保持される。そして、ログ書き込み手段３８により、ログバッファ３７内の情報がログボリューム３２に書き込まれる。その際、シーケンス番号採番手段３０ｂによりシーケンス番号が昇順で採番され、書き込むべきログに付与される。また、ファイルシステム復元手段３９により、ログボリューム３２に格納されたログを用いてメタボリューム３１内のメタデータの不整合が修正されると、修正した時に用いられたログの最後のシーケンス番号が初期シーケンス番号記憶手段３０ａに記憶される。その後、ログ書き込み手段３８により、ログバッファ３７内の情報がログボリューム３２に書き込まれると、シーケンス番号採番手段３０ｂにより、初期シーケンス番号採番手段３０ａに記憶されたシーケンス番号を基準としてシーケンス番号が昇順で採番され、書き込むべきログに付与される。

【００５４】これにより、既にファイルシステムの復元に用いたログと、ファイルシステム復元後に介したトランザクションのログとのシーケンス番号が重ならないことを保証することができる。その結果、システムが使用可能な期間中にログボリュームをゼロクリアする必要がなくなり、ゼロクリアに伴う処理の遅延を避けることができる。

【００５５】図４は、第４の発明の原理構成図である。第４の発明は、同じメタデータに対して複数回更新処理が行われる場合に、最終形態のメタデータのみをログとして採取するものである。第４の発明は、以下のような要素で構成される。

【００５６】メタボリューム４１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。メタキャッシュ４２は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段４３は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ４２へと読み込む。トランザクション４４は、メタキャッシュ４２内に読み込まれたメタデータの内容を更新する。ログ採取手段４５は、トランザクション４４の種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、メタキャッシュ４２内で変更されたメタデータの最終形態のみをログとして採取する。ログバッファ４６は、ログ採取手段４５が採取した情報を保持する。

【００５７】このようなデータ処理装置によれば、トランザクション４４がメタデータの更新をする際には、まず、メタデータ読み込み手段４３によりメタボリューム４１内の必要なメタデータがメタキャッシュ４２に読み込まれる。ここで、トランザクション４４がメタデータの内容を更新する。すると、ログ採取手段４５により、トランザクション４４の種別が判断され、メタデータの

更新を複数回行う可能性のあるメタデータ属性を予測する。予測されたメタデータ属性において、同一メタデータが何度も更新される可能性がある場合には、その属性のメタデータがメタキャッシュ４２内で変更された時点ではログ採取を行わず、トランザクション４４が終了した時点で、最終形態のメタデータをログとして採取する。採取した情報は、ログバッファ４６に保持される。

【００５８】これにより、複数回更新されたメタデータのログを更新処理の度に採取することがなくなり、メモリの効率化を図ることができるとともに、ログを書き出すときのＩ／Ｏの量も減らすことができる。

【００５９】図５は、第５の発明の原理構成図である。第５の発明は、メタデータ割り当て管理情報の一部の複製を生成し、複製として生成した情報内からのみメタデータの獲得を可能とし、解放する際には、割り当て管理情報においてのみ解放された旨の情報の更新を行うことで、解放直後に別のトランザクションに獲得されるのを防止したものである。第５の発明は、以下のような要素で構成される。

【００６０】割り当て管理情報保持手段５１は、メタデータに対する割り当てを管理するための割り当て管理情報５１ａを複数の領域に分割して保持する。獲得操作用管理情報生成手段５２は、割り当て管理情報保持手段５１内の割り当て管理情報の一部領域の複製を生成し、獲得操作用管理情報５１ｂとする。また、獲得操作用管理情報５１ｂ内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を獲得操作用管理情報５１ｂとする。トランザクション５３は、メタデータの獲得及び解放要求を出力する。メタデータ獲得手段５４は、トランザクション５３によりメタデータの獲得要求が出された場合には、獲得操作用管理情報５１ｂの中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように獲得操作用管理情報５１ｂと割り当て管理情報５１ａとの内容を変更する。メタデータ解放手段５５は、トランザクション５３によりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、割り当て管理情報の内容を変更する。

【００６１】このようなデータ処理装置によれば、獲得操作用管理情報生成手段５２により、割り当て管理情報５１ａの一部領域の複製が生成され、獲得操作用管理情報５１ｂとされる。トランザクション５３により獲得要求があると、メタデータ獲得手段５４により、獲得操作用管理情報５１ｂ内の未獲得のメタデータが獲得される。すると、獲得操作用管理情報５１ｂと割り当て管理情報５１ａとの内容が更新される。また、トランザクション５３よりメタデータの解放要求があると、メタデータ解放手段５５により該当するメタデータの解放処理が行われる。この際、割り当て管理情報５１ａの内容のみが更新される。ここで、獲得操作用管理情報５１ｂ内に未獲得のメタデータがなくなると、割り当て管理情報５１ａ内の別の領域の複製が獲得操作用管理情報５１ｂとされる。

【００６２】これにより、解放された旨の情報が獲得操作用管理情報５１ｂに反映されないため、解放直後のメタデータが他のトランザクションに獲得されることがなくなる。その結果、解放処理を行ったトランザクションの終了前にシステムがダウンしても、少なくとも解放前の状態のまま保全されることが保証される。

【００６３】図６は、第６の発明の原理構成図である。第６の発明は、トランザクション単位に、獲得や解放に関する情報をログとして記録することで、必要なメモリ容量の削減を図るとともに、平行動作するトランザクションのログに起因して割り当て管理情報に不正な状態が発生することを防ぐものである。第６の発明は、以下のような要素で構成される。

【００６４】割り当て管理情報保持手段６１は、メタデータに対する割り当てを管理するための割り当て管理情報を保持する。トランザクション６２は、メタデータの獲得及び解放要求を出力する。メタデータ獲得手段６３は、トランザクション６２によりメタデータの獲得要求が出された場合には、獲得操作用管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように割り当て管理情報６１ａの内容を変更する。メタデータ解放手段６４は、トランザクション６２によりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、割り当て管理情報６１ａの内容を変更する。ログ採取手段６５は、割り当て管理情報内のメタデータ獲得手段６３及びメタデータ解放手段６４によって変更された部分の情報をログとして採取する。ログバッファ６６は、ログ採取手段６５が採取したログを保持する。

【００６５】このようなデータ処理装置によれば、トランザクション６２からメタデータの獲得要求が出されると、メタデータ獲得手段６３によって未獲得のメタデータの１つが割り当て管理情報６１ａ内から獲得され、そのメタデータが獲得済みの状態とされる。また、トランザクション６２からメタデータの解放要求が出されると、メタデータ解放手段６４によって該当するメタデータが未獲得の状態に変更される。そして、ログ採取手段６５により、割り当て管理情報６１ａ内のメタデータ獲得手段６３及びメタデータ解放手段６４によって変更された部分の情報がログとして採取され、ログバッファ６６に保持される。

【００６６】このように、獲得、解放のログを採取する際に、トランザクションが獲得や解放を行ったという情報のみをログとして格納することで、メモリ等の領域を効率よく利用することができるとともに、他トランザクションによる獲得や解放処理の情報がログに含まれないことによって、割り当て管理情報が不正な状態となるこ

とを防ぐことができる。

【００６７】図７は、第７の発明の原理構成図である。第７の発明は、ログバッファを複数設けることにより、トランザクションの独立性をいっそう高めたものである。第７の発明の構成要素は以下の通りである。

【００６８】メタボリューム７１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。メタキャッシュ７２は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段７３は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ７２へと読み込む。複数のトランザクション７４ａ～７４ｃは、メタキャッシュ７２内に読み込まれたメタデータの内容を更新する。複数のログバッファ７５ａ～７５ｅは、ログをトランザクション毎に保持する。各ログバッファ７５ａ～７５ｅのサイズは一定ではなく、大きなサイズや小さなサイズが存在する。ログ採取手段７６は、メタキャッシュ７２内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて適したサイズのログバッファ７５ａ～７５ｅに格納する。例えば、最初にログを格納する場合には、トランザクションの内容によって予想される処理に適した大きさのログバッファに格納し、格納対象となるログバッファの記憶容量が不足してきたら、より大きな記憶容量のログバッファへログを移し替え、以後、より大きな記憶容量のログバッファをログの格納対象とする。

【００６９】このようなデータ処理装置によれば、トランザクション７４ａ～７４ｃがメタデータの更新をする際には、まず、メタデータ読み込み手段７３によりメタボリューム７１内の必要なメタデータがメタキャッシュ７２に読み込まれる。ここで、トランザクション７４ａ～７４ｃがメタデータの内容を更新する。すると、ログ採取手段７６により、メタキャッシュ７２内で変更されたメタデータがログとして採取され、適したサイズのログバッファ７５ａ～７５ｅに保持される。

【００７０】このように、複数のログバッファに分け、トランザクション毎に１つのログバッファを使用するようにしたことで、トランザクションの独立性を保つことができる。しかも、複数のサイズのログバッファを用意し、トランザクション毎に適したサイズのログバッファを使用することにより、メモリを効率的に利用できる。

【００７１】図８は、第８の発明の原理構成図である。第８の発明は、あるトランザクションのログがログバッファに入りきらない場合に、中間ログとしてログバッファの内容を書き出すものである。第８の発明の構成は以下の通りである。

【００７２】メタボリューム８１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム８２は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ８３は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段８４は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ８３へと読み込む。トランザクション８５は、メタキャッシュ８３内に読み込まれたメタデータの内容を更新する。ログバッファ８６は、ログをトランザクション毎に保持する。ログ採取手段８７は、メタキャッシュ８３内で変更されたメタデータの内容をログとして採取し、ログバッファ８６に格納する。ログ書き込み手段８８は、トランザクション８５が終了した場合にログバッファ８６の内容をログボリューム８２に書き込むとともに、トランザクション８５によるログがログバッファ８６内に格納しきれない場合には、ログバッファ８６内のデータを完結したログに加工し、中間ログとしてログボリューム８２内に格納する。なお、中間ログを生成する際には、中間ログに対してトランザクションを実行するのに必要とされたパラメタに関する情報を付加する。ファイルシステム復元手段８９は、ファイルシステム復元要求を受け取ると、ログボリューム８２に格納されたログを用いて、メタボリューム８１内のメタデータの不整合を修正する。このとき、中間ログを発見すると、中間ログに含まれたパラメタを用いてトランザクションを再実行させる。

【００７３】このようなデータ処理装置によれば、トランザクション８５がメタデータの更新をする際には、まず、メタデータ読み込み手段８４によりメタボリューム８１内の必要なメタデータがメタキャッシュ８３に読み込まれる。ここで、トランザクション８５がメタデータの内容を更新すると、ログ採取手段８７によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ８６に保持される。そして、ログバッファ８６に格納しきれなくなるか、トランザクション８５が終了すると、ログ書き込み手段８８により、ログバッファ８６内の情報がログボリューム８２に書き込まれる。ログバッファ８６に格納しきれなくなった場合には、ログバッファ８６の内容を完結したログに加工し、中間ログとしてログボリューム８２に格納する。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段８９により、ログボリューム８２に格納されたログを用いて、メタボリューム８１内のメタデータの不整合が修正されるとともに、中間ログまで採取した段階で停止しているトランザクションが再実行される。

【００７４】これにより、メタデータの更新を大量に行うトランザクションの実行中にシステムがダウンした場合には、途中の状態まで戻すことができるとともに、トランザクションが再実行されることで、トランザクションの実行後の状態へ遷移させることができる。

【００７５】図９は、第９の発明の原理構成図である。トランザクションの受け入れを一定の条件によって制限

することで、メモリ枯渇等を防止するものである。第９の発明の構成は以下の通りである。

【００７６】メタボリューム９１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム９２は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ９４は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段９３は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ９４へと読み込む。互いの同時実行可能な複数のトランザクション９０ｂ～９０ｄは、メタキャッシュ９４内に読み込まれたメタデータの内容を更新する。トランザクション受け入れ制限手段９０ａは、トランザクション９０ｂ～９０ｄからの開始要求を受け付けると、ログ採取に関するシステムの動作状況に基づいて、トランザクション９０ｂ～９０ｄの受け入れ許否を判断する。受け入れ判断基準としては、例えばログボリューム内の有効なログが占める割合を用いる。すなわち、有効範囲監視手段９９によって有効なログとされたログがログボリューム中に占める割合が一定値以上である間は、トランザクション９０ｂ～９０ｄの受け入れを拒絶する。

【００７７】ログ採取手段９６は、メタキャッシュ９４内で変更されたメタデータの内容をログとして採取する。ログバッファ９５は、ログ採取手段９６が採取した情報を保持する。ログ書き込み手段９７は、ログバッファ９５が保持する情報を、適宜ログボリューム９２に格納する。メタデータ書き込み手段９８は、メタキャッシュ９４内のメタデータをメタボリューム９１内に格納する。有効範囲監視手段９９は、メタデータ書き込み手段９８による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応するログボリューム９２内のログを、有効なログとして指定する。

【００７８】このようなデータ処理装置によれば、トランザクション９０ｂ～９０ｄから開始要求が出されると、トランザクション受け入れ制限手段９０ａが受け入れの許否を判断する。例えば、有効範囲監視手段９９により有効であると指定されたログのログボリューム９２内に示す割合が一定以上の場合には、それ以上ログが発生しないように、トランザクションの受け入れを拒絶する。その後、メタデータ書き込み手段９８によるメタデータの書き込みが進み、有効なログの割合が減少したら、その時点でトランザクションの開始要求を許可する。

【００７９】これにより、ログボリュームの空き容量の減少に伴うハングアップなどの障害の発生を防止することができる。次に、本発明の実施の形態を具体的に説明する。

【００８０】図１０は、本発明を適用するデータ処理装置のハードウェア構成図である。データ処理システムは、ＣＰＵ(Central Processing Unit)２１１を中心に構成されている。ＣＰＵ２１１は、バス２１７を介して他の機器を制御するとともに、様々なデータ処理を行う。バス２１７には、メモリ２１２、入力機器インタフェース２１３、表示制御回路２１４、ＨＤＤ(Hard Disk Drive)インタフェース２１５、及びネットワークインタフェース２１６が接続されている。

【００８１】メモリ２１２は、ＣＰＵ２１１が実行すべきプログラムや、プログラムの実行に必要な各種データを一時的に保持する。入力機器インタフェース２１３は、入力機器としてキーボード２２１とマウス２２２が接続されており、これらの入力機器からの入力内容をＣＰＵ２１１に伝える。

【００８２】表示制御回路２１４は、表示装置２２３が接続されており、ＣＰＵ２１１から送られてきた画像データを表示装置２２３で表示可能な画像情報に変換し、表示装置２２３の画面に表示させる。

【００８３】ＨＤＤインタフェース２１５は、複数のＨＤＤ２３１～２３３が接続されており、ＣＰＵ２１１から送られてきたデータをＨＤＤ２３１～２３３に格納するとともに、ＣＰＵ２１１からの要求に応じてＨＤＤ２３１～２３３内のデータを読み取り、ＣＰＵ２１１に転送する。

【００８４】ネットワークインタフェース２１６は、ＬＡＮ(Local Area Network)に接続されており、ＬＡＮを介してデータ通信を行う。すなわち、ＣＰＵ２１１から送られたデータをＬＡＮに接続された他のコンピュータに転送するとともに、他のコンピュータからＬＡＮを介して送られてきたデータをＣＰＵ２１１に転送する。

【００８５】ＨＤＤ２３１～２３３には、各種ファイルや、そのファイルを管理するためのメタデータ及びログが格納されている。このような構成のシステムにおいて、ＣＰＵ２１１がＨＤＤ２３１～２３３に格納されたオペレーティングシステム用のプログラムを実行することにより、本発明のログ採取機能が実現される。

【００８６】図１１は、ファイルシステム上で動作するログ採取機能の構成図である。図のように、ファイルを管理するためのメタボリューム１１１～１１３も複数設けられている。メタボリューム１１１～１１３には、それぞれメタデータが格納されている。メタデータは、ファイルの格納場所等を管理するために必要な情報を有している。

【００８７】ログボリューム１２０は、ログ１２２を格納するための二次記憶装置である。ログボリューム１２０には、ログ１２２の他にボリューム管理情報１２１が格納されている。

【００８８】メタキャッシュ１３０は、メタデータを操作するためのメモリ上の領域である。メタキャッシュ１３０内には、操作対象となるメタデータ１３２とそのメ

タデータの割り当て管理情報１３１とが格納される。

【００８９】ログキャッシュ１４０は、複数のログバッファ１４１～１４４を有している。ログバッファ１４１～１４４のサイズは均一ではなく、大きなサイズのものや小さなサイズのものがある。これらのログバッファ１４１～１４４には、メタキャッシュ１３０内で更新されたメタデータの複製がログとして格納される。

【００９０】ログキャッシュ１４０とは別にログライトバッファ１５０が設けられている。ログライトバッファ１５０には、トランザクションの処理が終了した時点で、ログバッファ１４１～１４４内のログが転送される。

【００９１】この例では、複数のトランザクション１０１～１０３が並列動作している。このトランザクション１０１～１０３は、ファイルシステムオペレーションを分割したものである。

【００９２】実際にＩ／Ｏを行うのは２つのデーモンであり、それぞれをメタライトデーモン１０４、ログライトデーモン１０５と呼ぶ。ログライトデーモン１０５がログをログ専用の二次記憶装置であるログボリューム１２０に出力する。メタライトデーモン１０４は、ログボリューム１２０に対してログが出力されたことを確認した後、そのログに対応するメタデータをメタデータ専用の二次記憶装置であるメタボリューム１１１～１１３に出力する。

【００９３】さらに、本発明のログ採取機能では、以下のような特徴を有している。第１の特徴は、各メタデータに対応するメタデータ管理情報として、メタボリュームを識別する情報が付加されていることである。これは、大規模ファイルシステムに対応するためのものである。すなわち、複数の二次記憶装置がメタボリューム１１１～１１３として定義される場合に、メタキャッシュ１３０上のメタデータ管理情報にそのメタデータが存在するボリュームの情報を持たせている。

【００９４】図１２は、メタデータ管理情報を示す図である。メタデータ管理情報には、「ボリューム番号」、「メタデータ番号」、及び「メタデータポインタ」が登録されている。「ボリューム番号」には、対応するメタデータが存在するボリューム番号が登録されている。「メタデータ番号」には、ボリューム毎におけるメタデータ管理番号が登録されている。すなわち、システムが認識するボリュームのデバイス番号とそのボリューム内の位置から算定される数値によってメタデータが管理され、メタデータ自体がそれらの値を管理情報として保持する。「メタデータポインタ」には、メタデータの実体のある場所を指し示している。

【００９５】このようなメタデータ管理情報を有するメタデータの内容が更新されると、更新後のメタデータの内容がログとしてログバッファに記録されるとともに、メタデータ管理情報の内容がログに記録される。

【００９６】図１３は、ログバッファの形式を示す図である。ログバッファには、「ＢＥＧＩＮマーク」、「ボリューム番号」、「メタデータ番号」、「メタデータ実体」、及び「ＥＮＤマーク」の情報を含んでいる。

【００９７】ファイルシステムを復元する際には、ログ内に記されているボリューム番号によって、そのログによって復元すべきメタデータの存在するメタボリュームが認識される。これにより、従来技術のようにファイルシステム復元時にメタデータ番号からその存在すべきボリュームを決定するよりも速度向上が望め、システムダウン後の大規模ファイルシステムにおいてもログ機構導入によるファイルシステム復元時間の短縮が有効に機能する。

【００９８】第２の特徴は、メタキャッシュ１３０内で更新されたメタデータが、それぞれの管理構造にリンクポインタを持つことである。ログとしてログボリューム１２０に記録されたメタデータはメタライトリストに繋がれ、メタボリュームへ反映が完了した時点でこのメタライトリストから外される。さらに、ログとして記録されているログボリューム１２０内の位置情報をも、メタライトリスト内に持つ。このログボリューム内位置情報をリストを辿って検索することによって、システムダウン時に必要とされるログの範囲を特定することができる。そこで、ここから得られる有効範囲情報をログボリューム１２０の特定位置に設けたボリューム管理情報１２１内に記録する。有効範囲情報を記録し、ファイルシステム復元時にそこに含まれるログのみをリプレイすることにより、ログボリューム全体を検索する必要がなくなり、ログボリューム全体を読み込む必要のある、有効範囲情報を記録しない従来の方式よりもファイルシステムの復元時間をさらに短縮することができる。

【００９９】ところで、ログ機構はシーケンシャル性を持ったディスクアクセスを行うことで、記録すべき内容をヘッドシークすることなしにディスクへ保存でき、ディスクアクセス時間の短縮を図っている。ところが、本手法を採用した場合、メタデータのメタボリューム反映時、ログボリュームへの書き出し時に、ログボリュームの特定位置に有効範囲情報を書き出すこととなる。これではシーク削減の意図が全く意味を成さない。

【０１００】そのため、本実施の形態ではある程度のインターバルを空けて、有効範囲情報は書き出すように工夫した。変更がある度に、常に書き込まれるわけではないため有効範囲情報として保存されている情報には若干の誤差が含まれてしまう。ファイルシステム復元時にその誤差を吸収する必要がある。

【０１０１】図１４は、有効範囲を説明する図である。図中において、「●」で示すのが、まだメタボリュームに反映が終了しておらず、ファイルシステム復元に必要なログを意味する。「○」で示すのは、メタボリュームに反映が完了したことによって、ファイルシステム復元

時には利用しなくても良いログである。

【０１０２】従来のファイルシステムにおいて、ログを用いたファイルシステム復元では、ログボリューム全体を検索し，ログに記録されたシーケンス番号から、最古のログを求め、たとえそれがメタボリュームに反映済みの、利用しなくても良いログであっても利用して、ログボリューム全体のログを用いていた。

【０１０３】本発明では、ある時点で有効範囲情報を書き出した時の有効範囲が示されている。その後、有効範囲を書き出さずに、メタボリュームへの反映が進み、また、別のトランザクションのログがログボリュームに書き出されたことによって、実際の有効範囲と有効範囲情報とはズレを生じている。この時点でシステムがダウンし、ログを用いてファイルシステムを復元する場合には、多少のムダが生じるが、有効範囲情報が示す先頭位置から、ログを利用する。利用すべきログの末端は有効範囲情報以降の一定範囲を検索しなければならないが、その検索は有効範囲情報を書き込むインターバルに依存し、範囲が限られている。

【０１０４】第３の特徴は、ログに対してトランザクション終了時にシーケンス番号を与え、その番号にはデータサイズが肥大化することを考慮に入れた上で、十分大きいデータ型を適用することである。データ型の大きさは、コンピュータ自身の使用可能年数の間使い続けても枯渇しない程度のサイズとする。例えば、システムの日時表記を４桁の十進数「１９９９」で表していた場合、西暦１万年まで使用されることは想定されていない。その場合、西暦９９９９年まで使用可能なデータ型とすれば、ログに対するシーケンス番号がオーバーフローして逆転することがないことを保証することができ、それにより、ログボリューム全体をゼロクリアして初期化する必要が生じない。具体的には、データ型を６４ビット型とすれば、オーバーフローすることは現実にはありえない（４万年ほど耐えられるものと思われる）。ファイルシステム管理情報、各トランザクションのログ、有効範囲情報にこのシーケンス番号を含め、ログボリュームに記録する。このように、ログに与えるシーケンス番号のデータ型に十分大きいものを適用することにより、通常の運用時にトランザクションが動作する毎にインクリメントしても、オーバーフローは現実には起こり得ない。

【０１０５】さらに、スーパブロックと呼ばれるファイルシステム全体の管理情報にこのシーケンス番号を含め、正常なアンマウント処理時及びファイルシステム復元時にスーパブロックに含まれるシーケンス番号を正しく設定し、次回のマウント時にそこから得られる値を用いる。これにより、シーケンス番号は必ず昇順となることが保証され、ファイルシステム復元時に新しいログを古いものと取り違えないことが保証される。

【０１０６】シーケンス番号の順序性を保証することによって、ファイルシステム復元時にログボリュームをゼロクリアしなくとも、常に正しいログを利用することが可能となり、ファイルシステム復元時にログボリューム全体をゼロクリアするのに比べ、大幅な時間短縮が可能となる。

【０１０７】以上の理由により、ログの最大のメリットであるファイルシステム復元の時間短縮がより一層有効に機能することが可能となる。第４の特徴は、トランザクションによって更新されたメタデータが、そのメタデータの属性に応じ、再度同一のトランザクションにおいて更新される可能性のあるメタデータであった場合には、更新の時点ではログバッファにはコピーせず、リスト構造（トランスリスト）によって管理することである。

【０１０８】そして、同一トランザクション内で複数回更新されるメタデータとして、領域割当て管理情報をリスト構造で管理し、トランザクション進行中にはその中途段階のログは採取しない。トランザクション終了時にリスト構造を辿って、トランザクションの更新の最終状態のみを一括してログとすることによって、ログの縮小が図られ、それに伴いファイルシステム復元時間の短縮が可能となる。

【０１０９】具体的には、メタキャッシュ１３０内のメタデータを管理する構造にトランスリストへのリンクポインタを持たせることによって実現している。トランザクションによって更新されたメタデータは、ただ一回しか更新されないことが分かっているメタデータである場合には、その時点でログバッファへコピーされるが、以降もトランザクション進行中に更新される可能性がある場合には、このリンクポインタを用いてリスト構造に繋がれる。メタデータの更新可能性の有無は、トランザクションの種別で判断する。例えば、データ領域の空きなどを管理するためのトランザクションでは、何度もメタデータが更新される。

【０１１０】そして、トランザクションが終了する時にこのトランスリストを辿り、メタデータの最終形態をログバッファへコピーする。すると、結局はトランザクションが同一メタデータを複数回更新しても、最終形態の一回だけのログ採取で済まされる。

【０１１１】図１５は、ログ採取処理のフローチャートである。この処理は、オペレーティングシステムを実行するＣＰＵが行う処理である。以下、ＣＰＵがオペレーティングシステムを実行することにより実現する機能を、単に「システム」ということとする。

［Ｓ１］トランザクションの開始宣言を行う。

［Ｓ２］ログ採取要求を行う。

［Ｓ３］対象メタデータの更新可能性の有無を判断する。更新可能性があればステップＳ５に進み、そうでなければステップＳ４に進む。

［Ｓ４］ログバッファへメタデータをコピーし、ステップＳ２に進む。

［Ｓ５］対象メタデータはトランスリストに繋がれているか否かを判断する。トランスリストに繋がれていればステップＳ７に進み、そうでなければステップＳ６に進む。

［Ｓ６］メタデータ毎にトランスリストに繋ぐ。

［Ｓ７］トランザクションの処理が終了するか否かを判断する。終了するのであればステップＳ８に進み、そうでなければステップＳ２に進む。

［Ｓ８］トランザクション終了宣言を行う。

［Ｓ９］トランスリストを辿り、繋がれている最終形態のメタデータをそれぞれログバッファへコピーする。

【０１１２】このようにして、トランザクションの更新の最終状態のみを一括してログとして保存することができる。第５の特徴は、メタボリュームの割当て管理情報は獲得用として通常の割当て管理情報の複製を新たに設けたことである。ここで、割り当て管理情報として、ビットマップを例に挙げて説明する。

【０１１３】図１６は、メタボリュームの割り当て管理状況を示す図である。割り当て管理情報１３１には、使用されているメタデータを管理するためのビットマップ１３１ａが用意されている。ビットマップ１３１ａは複数のブロックに分けられている。そして各ブロックのビットマップの各ビットが「０」か「１」かによって、対応するメタデータが空いているか否かが示される。そして、ブロックに分けられたビットマップの１つの複製が作られ、獲得用ビットマップ１３１ｂとされる。

【０１１４】獲得時には、通常のビットマップ操作と同様に未割り当て状態の領域の検索が行われる。この時、検索対象として用いるのは獲得用ビットマップ１３１ｂである。獲得用ビットマップ１３１ｂ内から未割り当て状態の領域（ビットが立っていない）が見つかった時には、その獲得要求には見つかったビット番号を返す。そして、獲得用ビットマップ１３１ｂ及び、その複製元となったビットマップの対応ビットを立てる。一方、獲得用ビットマップ１３１ｂ内から未割り当て状態の領域が見つからなかった時には、別のブロックのビットマップの複製を生成し、新たな獲得用ビットマップ１３１ｃとする。

【０１１５】図１７は、ビットマップによるメタデータ獲得処理を示すフローチャートである。この処理は、システムが行う処理である。

［Ｓ１１］獲得要求を発行する。

［Ｓ１２］獲得用ビットマップに空きがあるか否かを判断する。空きがあればステップＳ２０に進み、そうでなければステップＳ１３に進む。

［Ｓ１３］メモリ（メタキャッシュ１３０）上のビットマップに「FreeDirty」フラグが立てられていないビットマップが存在するか否かを判断する。存在すればステップＳ１４に進み、存在しなければステップＳ１６に進む。ここで「FreeDirty」フラグとは、一回以上の解放処理が対象ビットマップに対してなされたことを意味する。

［Ｓ１４］「FreeDirty」フラグが立てられていないビットマップの中で、空きのあるものがあるか否かを判断する。そのようなビットマップがあればステップＳ１５に進み、そうでなければステップＳ１６に進む。

［Ｓ１５］「FreeDirty」フラグが立てられておらず、空きのあるビットマップの複製を、獲得用ビットマップに作成する。その後、ステップＳ２０に進む。

［Ｓ１６］メタボリューム１１１～１１３上のビットマップに空きがあるか否かを判断する。空きのあるビットマップがあればステップＳ１７に進み、そうでなければステップＳ１８に進む。

［Ｓ１７］メタボリューム１１１～１１３上の空きのビットマップをメモリ（メタキャッシュ１３０）上に読み込み、ステップＳ１５に進む。

［Ｓ１８］メモリ（メタキャッシュ１３０）上のビットマップに「FreeDirty」フラグが立てられたビットマップが存在するか否かを判断する。存在すればステップＳ１９に進み、存在しなければ、獲得不可能と判断し処理を終了する。

［Ｓ１９］「FreeDirty」フラグが立てられたビットマップをメタボリュームに反映し、「clean」状態にする。ここで「clean」状態とは、獲得、解放の処理が全く行われていない状態を示す。その後、ステップＳ１３に進む。

［Ｓ２０］獲得用ビットマップのビットを立てる。

［Ｓ２１］複製元ビットマップの同一ビットを立てる。

［Ｓ２２］複製元ビットマップに「AllocDirty」フラグを立てる。ここで、「AllocDirty」フラグとは、対象ビットマップから一回以上の獲得処理によって更新がなされた場合に、そのビットマップの管理構造体内のフラグに立てる値である。「AllocDirty」フラグや「FreeDirty」フラグが立ったビットマップはログ採取対象である。メタボリュームに反映された時にこれらのフラグは落とされ、「clean」状態となる。

【０１１６】このようにして、空きのビットマップを獲得できる。解放時にも、通常のビットマップ操作と同様に、要求された領域が対応するビットの算出がまず行われるが、対応するビットを落とす操作は、対象のビットマップに対してのみ行い、仮にその対象ビットマップの複製が獲得用ビットマップとして存在する場合にも、その獲得用ビットマップに対しては行わない。

【０１１７】図１８は、解放処理のフローチャートである。この処理は、システムが行う。

［Ｓ３１］解放要求を出す。

［Ｓ３２］対象ビットマップがメモリ（メタキャッシュ１３０）上にあるか否かを判断する。対象ビットマップがあればステップＳ３４に進み、そうでなければステップＳ３３に進む。

［Ｓ３３］メタボリューム１１１～１１３上の対象ビットマップをメモリ（メタキャッシュ１３０）上に読み込む。

［Ｓ３４］対象ビットマップの対応するビットを落とす。

［Ｓ３５］対象ビットマップに「FreeDirty」フラグを立てる。

【０１１８】この一見複雑に見える作業によって、解放した領域を即座に別の用途に利用してしまうことを回避することが可能となり、システムダウン時に中途までしか終了していなかった解放トランザクションが解放したはずである領域は、解放される直前の状態のまま保全されることが保証できる。

【０１１９】例えば、Ａというトランザクションが解放処理、Ｂというトランザクションが獲得処理を行う場合を考える。ここで、トランザクションＡの解放処理はトランザクションＢの獲得処理よりも先に行われ、かつ、トランザクションＡはトランザクションＢよりも後に終わる場合を例に挙げる。

【０１２０】図１９は、トランザクションの処理の獲得と終了の状況を示す図である。この図において、各トランザクションは「ＢＥＧＩＮ」で始まり、「ＥＮＤ」で終わることを意味し、トランザクションの「○」が解放処理を、「●」が獲得処理を意味する。

【０１２１】上図のように、トランザクションＡの解放処理がトランザクションＢの獲得処理よりも先に行われ、かつ、トランザクションＢのほうがトランザクションＡよりも先に終了した場合に従来のログ採取方式では問題が生ずることは既に述べた。

【０１２２】本発明が提案する手法を用いれば、トランザクションＢが獲得する領域がトランザクションＡが解放した領域となることは基本的にはなく、領域枯渇状態に近く、どうしてもこの領域を獲得せねばならない時には、一旦、該当ビットマップをメタボリュームに反映した後、獲得処理が行われる。これにより、ファイルシステム復元時に必要とされる、メタボリュームに未反映なメタデータを対象とするログを用いた復元では、管理情報上、二重獲得と見えるようなログは残り得ない。

【０１２３】また、トランザクションＢのログにはトランザクションＡの解放処理が記録されないため、トランザクションＢのログよりも後に復元に用いるトランザクションＡのログによって、トランザクションＢが獲得した領域を変更されることもありえない。

【０１２４】第６の特徴は、前述のようにログ機構の導入にあたり、獲得・解放操作のログとして、その獲得・解放した対象の情報（このビットを立てた・落した）のみを記録することである。これによって、複数の獲得・解放要求に対して、要求の発生後、トランザクションの終了時点までをシリアライズすることなくログ管理でき、かつログ採取量は減少し、複数トランザクション動作による変更をメタボリュームへ反映する回数をも減少させることができる。

【０１２５】第７の特徴は、ログキャッシュ１４０を複数のログバッファで管理し、この時、いくつかの異なるサイズとすることである。ログキャッシュ１４０を分割して複数のログバッファとして管理することによって、トランザクションの独立性を確立することが容易となるとともに、有限なメモリ空間を有効に活用することが可能となる。

【０１２６】まず、トランザクションがメタデータを更新するより以前に、自分がメタデータを更新する旨、システムに宣言する。この時、トランザクションの種別より予測されるログ採取情報に応じて、最適なサイズのログバッファがシステムより与えられ、トランザクション進行に伴い蓄積されるログはここへ溜められることとなる。上記第６の特徴で説明した手法に加え、トランザクション毎に異なるログバッファを用いることによって、複数のトランザクション更新の混じらない、純粋なログとして残すことが可能である。

【０１２７】第８の特徴は、上記第７の特徴で説明した手法に加え、トランザクションの開始時に予測したログ採取量よりも多くのログを採取しなければならなかった場合に、与えられたログバッファから、さらに大きいログバッファへ移行することである。システムの状況に応じて、トランザクションによって残されるログのサイズは変動し、その差異を複雑な処理を必要とせず、かつ、有限なメモリ空間を有効に活用して吸収することが可能である。

【０１２８】第９の特徴は、ログバッファに入りきらない場合には、ログボリュームへ中間ログを書き出すとともに、中間ログを書き出すことによる矛盾の発生を防止する機能を備えたことである。

【０１２９】ログボリュームに対するＩ／Ｏ発行を減少させるために、ログライトバッファと呼ぶ二次キャッシュ領域を設けている。終了したトランザクションのログは、ログバッファからこのログライトバッファへ移され、適時にログライトデーモン１０５によってログボリュームへ書き出される。この適時とは、負荷が低い場合には一定の周期で良いが、トランザクションによるメタデータの更新量が多く、最大のログバッファでも収まらない場合などには強制的なＩ／Ｏ発行が必要とされる。このような場合において、中途の段階でのログの出力では、まだ更新されていない（かもしれない）メタデータを含めた書き出しをファイルシステムの整合性を保つためには必要である。

【０１３０】このように有限なメモリ空間を有効活用することを考慮した上で、ログバッファ内に収まりきらないログをトランザクション途中で出力する時に、残されるログによって復元されるファイルシステムの整合性を正当なものとするための手法を提案している。

【０１３１】ファイルシステムに対する負荷がそれほど高くない場合には、ログのログボリュームに対するＩ／Ｏを極力少なくするために事前に与えられた周期に応じて定期的にデーモンによって行われる。しかし、ログバッファが枯渇し、以降のトランザクション進行で溜められるログが収まらないと判断された時、部分的なログとして、この時点のログをログボリュームへ出力する。この時、まだ更新されていない情報をもログに出力する。例えばファイルを追加ライトするトランザクションの場合は、部分的なログを出力する時点でのファイルサイズや時刻情報を合わせてログに記録する。これによって、部分的であるはずのこのログを、ファイルシステム復元時には完結した１つのトランザクションのログと同等に扱うことが可能となり、ファイルシステム復元時に複雑な考慮の必要なしに、望ましい状態にファイルシステムを復元することが可能となる。

【０１３２】図２０は、ログバッファへのログの格納状況を示す図である。この例では、２種類のサイズのログバッファ１４１～１４６が設けられている。ログバッファ１４１～１４５は通常のサイズであり、ログバッファ１４６が大きなサイズである。

【０１３３】また、ログバッファ１４１～１４６を管理するためのログバッファ管理テーブル１４８，１４９が設けられている。ログバッファ管理テーブル１４８は、ログバッファ１４１～１４６に対応するフラグビットを有している。そして、使用されているログバッファに対応するフラグビットの値が「１」に設定され、未使用のログバッファに対応するフラグビットの値は「０」に設定されている。同様に、大きいサイズのログバッファ１４６に対応するログバッファ管理テーブル１４９もフラグビットを有しており、そのフラグビットの値によって、ログバッファ１４６が使用されているか否かを管理している。

【０１３４】ログキャッシュ１４０内の各ログバッファ１４１～１４５のうち、ログバッファ１４３とログバッファ１４５とが現在利用中であることを意味するために、「●▲」などの記号を用いた。ここで、ログバッファ１４５には多くのログが溜まっており、既に満杯の状態となっている。このように通常のサイズのログバッファ１４１～１４５では容量が足りなくなった際には、そのログを大きいサイズのログバッファ１４６に移動する。そして、トランザクションが継続している間は、更新されたメタデータの内容を、随時ログバッファに格納していく。ここで、トランザクションが終了したら、そのトランザクションに対応するログバッファの内容をログライトバッファ１５０へ転送する。

【０１３５】ログライトバッファ１５０には、複数のバッファ１５１，１５２が設けられており、それらのバッファ１５１，１５２にログが書き込まれる。ログライトバッファ１５０内のログは、ログライトデーモン１０５によって随時ログボリュームに書き込まれる。

【０１３６】図２１は、ログ採取手順を示すフローチャートである。これはシステムが行う処理である。

［Ｓ４１］BEGIN宣言を行う。

［Ｓ４２］ログバッファの予約をする。

［Ｓ４３］ログ採取要求を行う。

［Ｓ４４］現在のログバッファに今回のログが収まるか否かを判断する。収まるのであればステップＳ５１に進み、収まらないのであればステップＳ４５に進む。

［Ｓ４５］現在のログバッファより大きいログバッファが存在するか否かを判断する。存在すればステップＳ４６に進み、そうでなければステップＳ４７に進む。

［Ｓ４６］現在のバッファから大きいバッファへ内容をコピーし、ステップＳ４４に進む。

［Ｓ４７］トランザクションに与えられたパラメタをログ採取する。

［Ｓ４８］現時点のファイル状態を正しく表す管理情報をログ採取する。

［Ｓ４９］現在の中途段階のログをログライトデーモン１０５に書き出させる。

［Ｓ５０］現在のログバッファをクリアする。

［Ｓ５１］ログを採取する。

［Ｓ５２］ログ採取要求を終了する。

［Ｓ５３］ＥＮＤ宣言あるか否かを判断する。END宣言があればステップＳ５４に進み、そうでなければステップＳ４３に進む。

［Ｓ５４］END宣言を行い、処理を終了する。

【０１３７】第１０の特徴は、トランザクションが完了する前に、システムダウンが発生した場合に備え、分割して出力するログにはトランザクションに与えられたパラメタを含め、ファイルシステム復元時にそのパラメタを元に、再度トランザクションを実行することである。

【０１３８】さらに、この部分的なログにトランザクションに与えられたパラメタを記録し、ファイルシステム復元時にそれを用いて、通常のログだけでは中途で終わった状態までしか復元できないファイルを、トランザクションが終了した状態にまでファイルシステムに反映することが可能となる。

【０１３９】図２２は、ファイルシステム復元処理を示すフローチャートである。これはシステムが行う処理である。

［Ｓ６１］システムダウン等が発生する。

［Ｓ６２］ファイルシステムの異常を検知する。

【０１４０】以降、ファイルシステム復元処理を行う。

［Ｓ６３］ログ開始マークを検出する。

［Ｓ６４］ログ開始マークに対応するログ終了マークが存在するか否かを判断する。ログ終了マークが存在すればステップＳ６５に進み、ログ終了マークが存在しなければステップＳ６６に進む。

［Ｓ６５］開始マークから終了マークまでのログを用い

て復元処理を実行する。その後、ステップＳ６３に進む。

［Ｓ６６］対象トランザクションが以前のログだけで完結するか否かを判断する。完結するのであれば復元処理を終了し、完結しないのであればステップＳ６７に進む。

［Ｓ６７］ログに記録されたパラメタを読み込む。

［Ｓ６８］トランザクションとして行いたかった処理を理解する。

［Ｓ６９］ファイルに対して直接処理を再実行する。その後、復元処理を終了する。

【０１４１】これにより、オペレーションのセマンティクスを保証したファイルシステムの復元が可能となり、従来技術では完全に元に戻す、または完全に再実行することが不可能であったトランザクションを、完全に再実行することによってファイルの整合性をも回復することが可能となる。

【０１４２】第１１の特徴は、メタデータを更新するトランザクションが更新処理を行う前に、その旨をシステムに対し宣言することである。さらに、システムは宣言を受け入れるか否かを判断する機構を持つことである。ここでは、以下の条件の場合には新規のトランザクションの受け入れを拒否し、拒否されたトランザクションは再度認可が下りるまで待合わせなければならない。

・メタキャッシュ１３０がフルに近い状態にある場合。

・トランザクションの多重度が、システムで規定された値を既に越えていた場合。

・ログボリュームに残されたログの大部分が有効なログ状態である場合。

【０１４３】これらの場合に、ファイルシステムが新規トランザクションの受け入れを拒否し、トランザクションの多重度を宣言することによって、結果としてダーティなメタデータの数を制限することが可能となり、メタキャッシュ１３０の空き領域枯渇などを要因とするハングアップ状態に陥ることを回避することが可能となる。特に、分割ログとしてログ採取しなければならないトランザクションが動作中に、新規トランザクションの開始を拒否することは、システムの状態を正常に維持する上で効果が高い。

【０１４４】第１２の特徴は、新規トランザクションの受け入れ拒否の機構をログボリューム内の有効ログの割合に応じて適用することにより、単一の、多数のメタデータを更新するトランザクションのみならず複数のトランザクションが更新したメタデータによって空き領域の減少に伴うハングアップ状態をも回避することである。

【０１４５】以下に、第１１の特徴と第１２の特徴とを含むトランザクションの受け入れ処理について説明する。図２３は、新規トランザクションの受け入れ許否判定処理を示すフローチャートである。

［Ｓ７１］トランザクションを行うプロセス（以下、単にトランザクションという）が、トランザクション処理を開始する。

［Ｓ７２］「BEGIN」宣言を行う。この際、オペレーティングシステム（以下、単に「システム」という）に対して問い合わせを行う。

［Ｓ７３］トランザクションは、システムからの回答待ち状態となる。システムより「ＯＫ」の回答を受け取ったらステップＳ８２に進む。

［Ｓ７４］トランザクションからの問い合わせを受けたシステムが、パラメタの評価を開始する。

［Ｓ７５］システムは、既に多重度が規定値より上であれるか否かを判断する。多重度が規定値より上であれば「ＮＧ」としてステップＳ８１に進み、そうでなければ「ＯＫ」としてステップＳ７６に進む。

［Ｓ７６］システムは、現在サブトランザクションが動作中であるか否かを判断する。サブトランザクションが動作中であれば「ＮＧ」としてステップＳ８１に進み、そうでなければ「ＯＫ」としてステップＳ７７に進む。ここでサブトランザクションとは、ログを複数に分割して格納する場合に、１つのログバッファにログを格納できるような処理単位に分割されたトランザクションである。

［Ｓ７７］システムは、メタキャッシュ１３０に十分な空きがあるか否かを判断する。十分な空きがなければ「ＮＧ」としてステップＳ７９に進み、十分な空きがあれば「ＯＫ」としてステップＳ７８に進む。

［Ｓ７８］システムは、ログボリュームに上書き可能領域が少ないか否かを判断する。上書き可能領域が十分になければ「ＮＧ」としてステップＳ７９に進み、十分な上書き可能領域があれば「ＯＫ」としてトランザクションに対して「ＯＫ」の回答を返す。

［Ｓ７９］システムは、ログライトデーモン１０５を起動する。

［Ｓ８０］システムは、メタライトデーモン１０４を起動する。

［Ｓ８１］システムは、ＮＧ条件が解消されるまでスリープ状態で待つ。ＮＧ条件が解消されたら、ステップＳ７５に進む。

［Ｓ８２］トランザクションは、システムからの「ＯＫ」の回答を受け取ったら、多重インクリメント処理を行う。

［Ｓ８３］トランザクションは、「BEGIN」処理を終了する。

［Ｓ８４］トランザクションは、処理に応じて、メタデータをキャッシュに読み込み、その度に空き量をデクリメントする。分割ログ化するなら他トランザクションの開始を拒否するようにシステムに依頼する。

［Ｓ８５］トランザクションは、「END宣言」を行う。

［Ｓ８６］トランザクションは、多重度をデクリメントする。

［Ｓ８７］トランザクションは、必要に応じてログ採取を繰り返した後、自己のトランザクション処理を終了する。

【０１４６】次に、本発明を適用したシステムの具体的な処理内容について説明する。本発明の全ての特徴を備えたシステムにおけるログ採取手順は以下のようになる。ファイルシステムオペレーションを分割したトランザクションはメタデータを更新するより前に、自分がこれからメタデータを更新する旨を宣言する（ＢＥＧＩＮ）。ＢＥＧＩＮ時にトランザクションに対し、独立したログバッファが割当てられる。この時、既にトランザクションの並列動作数が非常に多かったりメタキャッシュ１３０域の利用率が高い場合には、システムによってＢＥＧＩＮ宣言が拒否される。ＢＥＧＩＮ宣言を拒否されたトランザクションはその許否理由が解消されるまで、待ち合わせなければならない。

【０１４７】更新が完了したメタデータは、ＢＥＧＩＮ時に割り当てられたログバッファへコピーされる。それと同時にメタデータ種類に応じたリストへ繋がれる。更新すべき全てのメタデータを更新し終わったトランザクションは、そこで完了を宣言する（ＥＮＤ）。ＥＮＤ時には、これまで溜めたログバッファの内容をログ専用の二次キャッシュであるログライトバッファへ移動し、さらに、まだログバッファへコピーされていなかったメタデータをログライトバッファへコピーする。

【０１４８】また、ＥＮＤ時にメタデータの種類に応じて繋がれていたリストから、そのトランザクションが更新した全てのメタデータを「ログ待ちリスト」へ繋ぎ替える。

【０１４９】以上で非同期要求のトランザクションは終了することができる。トランザクションが同期要求であった場合には、ログを書き出すまで待ち合わせなければならない。

【０１５０】ログの書き出しは独立したデーモン（ログライトデーモン１０５）が行う。このデーモンの起動契機は、同期要求トランザクションによる起動要求（ｗａｋｅｕｐ）、メタキャッシュ１３０の空き状態監視機構による起動要求（ｗａｋｅｕｐ）、及びタイマである。

【０１５１】起動したログライトデーモン１０５は複数のトランザクションのログがまとめられたログライトバッファを１つのＩ／Ｏとして発行する。そして、ログ待ちリストから「書き出しリスト」へメタデータを移動する。

【０１５２】メタボリュームへＩ／Ｏを発行するのは、メタライトデーモン１０４の役割である。メタライトデーモン１０４は書き出しリストに繋がれたメタデータを順に非同期ライトによってメタボリュームに反映する。メタライトデーモン１０４の起動契機は、ログバッファの空きが少なくなった時、ログボリュームの空きが少なくなった時、及びタイマである。

【０１５３】以上で、メタデータを更新するトランザクションの開始から、更新されたメタデータがメタボリュームに反映されるまでの大まかな流れである。以降に、ログの採取とそのディスク反映手法、そして、メタデータのディスク反映の処理について詳細に説明する。

（１）ログの構造

（１．１）ログボリューム

ログボリュームに格納される情報は、以下のような情報である。

【０１５４】ログボリュームにはスーパブロック、ボリューム管理情報が先頭にある。その次にログの有効範囲を示す構造体を記録する。構造体は以下のメンバを持つ。

・有効ログ先頭のログボリューム内オフセット
・有効ログ先頭のログシーケンス番号
・有効ログ末尾のログボリューム内オフセット
・有効ログ末尾のログシーケンス番号

ただし注意しなければならないのは、ここで先頭・末尾と言っているのは、真の値ではない可能性があることをリプレイ時に考慮しなければならない点である。なぜなら、ログはボリュームをシーケンシャルアクセスすることによってシーク時間の短縮を計っているが、このようにボリュームの一部へ毎回アクセスしたのでは、そのシーケンシャル性が損なわれてしまう。そのために、ログの書き出し毎にはこの有効範囲の書き出しは行わず、数回に一回、書き出している。

【０１５５】これにより、上記構造体に収められたオフセットは正確ではないためにリプレイ時に検索が必要ではある。しかしながら、全体を検索するよりも大幅に検索量が減るため、利点は保持できるものと考える。

【０１５６】上記以外は、全て、メタデータの更新履歴によって構成される。

（１．２）ログブロックの基本構造

ログボリューム１２０内に残されたログ（メタデータの更新情報）は基本的にはトランザクション単位である。これをログブロックと呼ぶ。しかし、キャッシュ域フルなどによって、１つのトランザクションを一度にまとめることができない場合は複数の分割ログに分ける。この分割ログをサブトランザクションログと呼ぶ。

【０１５７】サブトランザクションログをリプレイすることによって、ファイルシステムの整合性が保てるよう、その内容は工夫されている。しかしながら、トランザクションの途中までのサブトランザクションをリプレイしたのでは、トランザクション全体が終わっていないことから、ファイルシステムとしての整合性が保てても、ファイルの中身は異常であるという状態に陥ってしまうことが考えられる。

【０１５８】この状態を回避するために、そのサブトランザクションに別れたトランザクションがどのようなオペレーションであったかをログに採取する（オペレーシ

ョンログ）。ログリプレイ時には、サブトランザクションをリプレイした後、オペレーションログをリプレイヤが再実行することによって、トランザクション全体が終わった状態とすることが可能である。

【０１５９】サブトランザクション化する可能性のあるトランザクションは全体終了時に「終わったこと」をログに記録し（最終ＥＮＤマーク）、リプレイヤはその有無を調べてオペレーションログの実行を判断する。

（１．３）ログの採取形式

メタデータの更新情報は該当するメタデータの管理構造単位で採取する。また、メタボリュームの管理構造であるビットマップはその更新部分だけのログ採取とする、スーパブロックについては特殊であり、データ空き容量についてのみログ採取を行う。

（１．３．１）ｉｎｏｄｅ、Ｖデータ、空き管理情報

ファイルの管理情報であるｉノード、Ｖデータと呼ぶディレクトリやシンボリックリンクデータや空き管理情報のメタデータ本体は、それらのＩ／Ｏ単位（ブロック単位）でのログ採取を行う。

【０１６０】これは、ファイルシステム整合性チェック処理であるｆｓｃｋによるログリプレイ時の処理を簡略化することを意識している。変更された部分だけをログ採取した場合、ログリプレイ時には、該当ブロックの算定→読み込み→変更部分の更新→再度書き込み、とステップが増えてしまう。しかし、ブロック全体のログ採取によって、リプレイ時にはメタボリュームにログ情報を上書きするだけでよい。

【０１６１】ｉノード本体やディレクトリブロックなどのＶデータは、更新中はファイルのロックで保護されている。しかしながら、空き管理情報についてはファイルのロックでは保護されない。そのため、トランザクション途中で他トランザクションによって更新され、そのトランザクションが追い越して先に終わってしまうと、ログには古い情報が後に残されてしまい、リプレイするとファイルシステムに不整合が生じてしまう。

【０１６２】そのため、空き管理部については、ここでは更新するトランザクションが並行動作しないことを保証することによって実質的な排他制御を行い、他のメタデータと同じ採取形式とした。

（１．３．２）ビットマップ

メタデータの割り当て状況を管理するビットマップも空き管理部と同様にファイルのロックで保護されていない。そのため、ビットマップ全体のログ採取を行うと、そのログには複数のトランザクションによる更新が含まれてしまう可能性がある。特に、トランザクションの追い越しが発生すると、新しいはずのログによって、ビットマップの情報が古く書き戻されてしまうことが考えられる。

【０１６３】そこで、ビットマップのログ採取は更新部分だけとする。更新部分とは、あるビットが０になった、あるいは１になったと記録するだけである。それにより、トランザクションが並行動作してもそれぞれのトランザクションが更新した内容のみがログに残されるため、メタデータの後退は起こり得ない。

【０１６４】注意すべきは、ある領域を解放（「１→０」）した後、他トランザクションがそこを獲得（「０→１」）した場合である。トランザクションの追い越しが発生すると、ログリプレイによって獲得した領域を解放してしまう。

【０１６５】これについては、アロケート用ビットマップを１つ定め、複製を用いて操作を行うことによって回避する。

（１．４）ログの記録構造

（１．４．１）一般ログ

有効状態であっても、ある程度の複数スレッドが同時に進行することを許す。これは性能要件より必須である。しかし、ログボリューム内のログは前述の通り、トランザクション毎に保存する。１つのトランザクション結果をログに記録する際、ログは以下のパーツで構成する。

１）ＢＥＧＩＮマーク
２）ヘッダ
３）更新内容
（２）＋３）の繰り返し）
４）ＥＮＤマーク

これを以下、ログブロックと呼ぶ。ログブロックはログボリュームの物理ブロック境界から始まるが、その終わりは物理ブロック境界であるとは限らない。

１）ＢＥＧＩＮマーク

トランザクションの開始時に作成される情報であり、以下の内容を含む。

・マジックワード
・トランザクションタイプ
・ログシーケンス番号
・ログブロックサイズ

マジックワード：１つのトランザクションが始まったことを示すマジックワードを埋め込む。

【０１６６】トランザクションタイプ：ＢＥＧＩＮマークにトランザクションのタイプ（種類）を埋め込むことによって、その後ろの更新情報に含まれる内容の概要を事前に知ることが可能。

【０１６７】ログシーケンス番号：ログに記録されたトランザクション毎にインクリメントされる数値。このカウンタが最小のログブロックがそのログボリューム内で最も古いトランザクションであることを示す。カウンタは６４ビット型とし、オーバーフローすることは現実にはありえない（４万年ほど耐えられるものと思われる）。ログリプレイ時に、ログボリュームをゼロクリアすることにより、再度０から開始することも考えられるが、リプレイ時間の短縮を考慮し、利用したログの最後のシーケンス番号より昇順に採番することによりゼロク

リアを回避する。

【０１６８】ログブロックサイズ：ログブロックのＥＮＤマークまでのサイズを記録する。ＢＥＧＩＮマークの先頭からＥＮＤマークの先頭までのサイズである。リプレイ時にはＢＥＩＧＮマークの先頭からこのサイズだけ移動し、ＥＮＤマークの情報を参照して、ログの有効性を判定する。

２）ヘッダ

更新されたメタデータについて、その保管位置を特定するための情報であり、以下のメンバによって構成される。

・メタデータタイプ

・メタボリューム番号

・ボリュームローカルメタデータ番号

メタデータタイプ：更新情報の後方にある又データ更新内容が何のメタデータであるのかを特定するために用いられ、リプレイヤはここから更新内容のサイズを判断する。

【０１６９】メタボリューム番号：メタデータ管理で用いているメタボリューム毎の番号を記録する。リプレイ時には、この番号から書き戻すメタボリュームを決定する。ボリュームローカルメタデータ番号：メタデータ管理で用いている、メタボリューム毎、メタデータ毎に０から始まる値を記録する。リプレイ時には、この番号からメタボリューム内のブロック位置に変換し、更新内容を書き戻す。これはブロック位置変換を削減することによるログ採取の高速化が狙いである。対象がビットマップである場合には、変更されたビットが該当するメタデータ本体のメタデータ番号を記載する（ビットマップ番号ではない）。

３）更新内容

メタデータ本体の場合には、更新内容が含まれるブロック（それぞれのメタデータの管理単位）である。例えば、ディレクトリブロックが更新された場合には１０２４バイトのディレクトリブロック全体である。

【０１７０】ビットマップの場合には、全体ではなく、ここには「０→１」または「１→０」という情報だけ記録する。リプレイ時にはヘッダから該当するビットマップを判定し、それを読み込み、ここの内容から該当ビットを変更して書き戻す。

４）ＥＮＤマーク

ＢＥＧＩＮマークに対応するマジックワード、ログシーケンス番号、ログブロックサイズを記録する。

【０１７１】ＥＮＤマークがマジックワードとログ番号だけでは、メタデータの内容によってはそれが偽りのＥＮＤマークとして見えてしまい、リプレイの時に処理を誤る可能性がある。これを回避するために、ＥＮＤマークは固有形式（メタデータを誤認することのない形式）としなければならない。

【０１７２】ＢＥＧＩＮマークを固有形式としないのは、ＢＥＧＩＮマークだけ書き出された時点で、システムダウンした場合を考慮し、いずれにせよＥＮＤマークの識別ができなければならないからである。

【０１７３】固有形式とするために、固有の数値を６４バイト分続ける。これにより、全てのメタデータがＥＮＤマークに化けることはなくなる。これに続けて、マジックワード、カウンタ（ログ番号）、ログブロックサイズを記録する。以降、ブロック境界まで、上記数値を埋める。ＢＥＧＩＮマークに対応したＥＮＤマークを見つけることができないログ情報はリプレイしてはならない（ログ書き出し途中にシステムが異常終了したことを意味する）。

（１．４．２）巨大トランザクションのログ

トランザクションが多くのメタデータを更新する場合には、ログバッファサイズ、ログボリュームサイズが有限であることから、複数のサブトランザクションログに分割する。サブトランザクションログはそれだけのリプレイによって、ファイルシステムの整合性は保たれる。しかし、トランザクション全体のサブトランザクションログをリプレイしなければ、ファイルの整合性が保てない。そのため、サブトランザクション途中でのシステムダウンを考慮し、オペレーションログ内部に含む。

１）ＢＥＧＩＮマーク

２）オペレーションログ

３）ヘッダ

４）更新内容

（３）＋４）の繰り返し）

５）ＥＮＤマーク

（１）～５）の繰り返し）

（５’）最終ＥＮＤマーク）

１）ＢＥＧＩＮマーク

一般ログのＢＥＧＩＮマークと同じ。ただし、サブトランザクションログとなりうるトランザクションは、書き込みや削除など、データ領域を触るものや、領域管理に関わるものに限定される。

【０１７４】これらのトランザクションがＢＥＧＩＮマーク内で設定されていたら、そのログはサブトランザクション化している可能性があり、必すオペレーションログが続いている（たとえ単一のサブトランザクションログで構成されていても）。

２）オペレーションログ

サブトランザクション化したトランザクション（関数）に与えられた引数をオペレーションログとして保存する。巨大トランザクションを対象としたＢＥＧＩＮマークの直後には必ずオペレーションログを配置する。オペレーションログは、最初にトランザクションへ渡された情報（このファイルをこれだけのサイズトランケートするとか、このファイルをこれだけアペンドライトするなど）である。

【０１７５】リプレイ時には、リプレイヤがファイルシ

ステムを直接操作して、ここで残されたオペレーションログを実行しなければならない。オペレーションログは以下のメンバで構成される。

・パラメタ１
・パラメタ２
・・・

リプレイヤはＢＥＧＩＮマーク内のトランザクションタイプを見ることによって、オペレーションログ部分に並ぶパラメタ群のサイズ及び内容を知ることができる。

３）ヘッダ

一般ログのヘッダと同じ。

４）更新内容

一般ログの更新内容と同じ。

５）５’）ＥＮＤマーク、最終ＥＮＤマーク

一般ログのＥＮＤマークと同じ意味合いを持ち、ＢＥＧＩＮマークに対応するＥＮＤマークあるいは最終ＥＮＤマークが見つからない場合には、ＢＥＧＩＮマーク以降の更新内容をリプレイしてはならない。

【０１７６】巨大トランザクションがサブトランザクションに分割されず、単一のログブロックのみの場合には、ＥＮＤマーク部分には最終ＥＮＤマークが書き出される。複数のサブトランザクションログに分かれている場合には、最後のログブロックだけ、ＥＮＤマークが最終ＥＮＤマークに化ける（最後以外のログブロックには通常のＥＮＤマークが書かれている）。

【０１７７】ＥＮＤマークは一般ログのＥＮＤマークと同一であり、マジックワード、カウンタ（ログ番号）、ログブロックサイズが記録されており、ブロック境界まで固有数値が埋まる。

【０１７８】最終ＥＮＤマークとＥＮＤマークの差は、マジックワードのみである。最終ＥＮＤマークがトランザクション全体の終了を表すことから、とても巨大なトランザクションが多くのサブトランザクションログに分割されている場合には、全ての処理が完了する時にこの最終ＥＮＤマークを出力する。すなわち、巨大トランザクションとして定義されるトランザクションについて、通常のＥＮＤマークで終わるログブロックがいくつか見つかったのに、最終ＥＮＤマークで終わるログブロックが存在しない場合には、それは巨大トランザクションの途中でシステムダウンが発生したことを意味し、ログリプレイヤがオペレーションログのリプレイを行わなければならないことを意味する。

（２）ログの採取

（２．１）ログバッファの管理

ログをトランザクション単位にログボリューム１２０へ出力するためには、メタデータの更新情報を一度、メモリ内に溜める必要がある。これをログバッファと呼ぶ。ログバッファはマウント時にまとめて用意し、ファイルシステム利用中にメモリの追加獲得は行わない。

【０１７９】トランザクション毎のログ採取とし、また、トランザクション動作中に他のトランザクションが並行動作することを狙い、ファイルシステム毎にログバッファを複数持つ。その数は並行動作を許すトランザクション数によって決められる。

【０１８０】並行動作数を限定することはログ機能追加による性能劣化要因の１つとなるが、

・ログリプレイ時の処理を単純化
・巨大なログバッファを分割して使うことによる複雑さの回避

などのメリットが考えられるため、この方式とする。

【０１８１】各トランザクションは開始時に割り振られたログバッファにログを作成し、トランザクション終了時にまとめてログライトバッファへコピーする。ログライトバッファの内容を実際にＩ／Ｏ出力するのはログライトデーモン１０５と呼ぶデーモンの働きによる。

【０１８２】巨大トランザクションには、一般トランザクションが用いるログバッファよりも大きいログバッファ（以降、それぞれを一般ログバッファ、巨大ログバッファと呼ぶ）を１つ用意する。巨大トランザクションも当初は一般ログバッファの１つを用いる。巨大トランザクションとログバッファの関係には次の二段階がある。

１）あまり多くのメタデータを更新せず、一般ログバッファで十分足りると判断できた時点で、次の巨大トランザクションのＢＥＧＩＮを受け付ける。

２）更新するメタデータ数がある程度多く、一般ログバッファでは足りないと判断できた時点で、巨大ログバッファにこれまでの内容をコピー、そこで処理を継続する。

３）更新するメタデータ数が大変多く、巨大ログバッファでは足りない場合には、サブトランザクションとして分割する。

【０１８３】２）におけるバッファ間のコピーが負担となりそうであるが、巨大トランザクションがそれほど多くないと考えると、あまり頻繁に発生するものではないので、この方法とする。

【０１８４】一般ログバッファは状態（利用中・未利用）について、ビットマップで管理される。各ログバッファを管理する構造をログバッファ数の配列で確保し、それぞれが

・ログバッファのアドレス
・これまでに採取したログサイズ

を持っている。

【０１８５】また、ログバッファの内容は専用のログライトデーモン１０５によってログボリューム１２０へ反映するが、そのログライトデーモン１０５が複数のログバッファの内容を一括してＩ／Ｏ発行するために、トランザクション終了時にそれぞれのログバッファの内容をログライトバッファ１５０と呼ぶ専用の領域にコピーする。

【０１８６】ログライトバッファを管理する構造とし

て、
・追加モードにあるログライトバッファはどちらか
・それぞれのログライトバッファに溜められたログサイズ
がある。

【０１８７】ログライトバッファの内容をログボリューム１２０へ書き出している間は、その後のトランザクションが終了するためにログライトバッファにログバッファをコピーしようとしても、１／０中であるためにガードしなければならない。これは性能劣化に繋がるため避けなければならない。そのため、ログライトバッファは２本用意する。

【０１８８】一般トランザクションはトランザクション開始時に、空のログバッファをビットマップより見つけ（未利用ログバッファを０で示し、ここで１とする）、そのビット番号がトランザクション番号となる。

【０１８９】巨大トランザクションが一般ログバッファにて処理進行中、一般ログバッファでは入りきらないと判断されるとこれまでの内容をこの巨大ログバッファへコピーし、そこで処理を継続する。この時、これまで用いていた一般ログバッファは空きバッファリストへ繋ぎ直される。

（２．２）巨大トランザクションの管理

空き領域管理ツリーや獲得済領域管理ツリーを操作するトランザクションを巨大トランザクションと定義する。巨大トランザクションは場合によっては木構造を大きく変更しなければならないかもしれない。この時、サブトランザクション化する必要が生ずる。

【０１９０】サブトランザクション化した巨大トランザクションが並行動作した場合、どれもが多くのメタデータを更新すると、メタキャッシュ１３０がいずれ全てダーティとなり、新たにメタデータを読み込んで更新しようにも追い出すこともできずにデッドロックに陥ることが考えられる。

【０１９１】そのため、巨大トランザクションについては並行動作ができないよう、ここではシリアライズした。しかし、上記の通り、多くのメタデータを更新する可能性があると考えられるトランザクションは多い。これらを全てトランザクション終了までシリアライズすると、その性能インパクトは大きい。

【０１９２】巨大トランザクションも、場合によってはそれほど多くのメタデータを更新しないこともありえる。一般トランザクションと同等の数しかメタデータを更新しないのであれば、扱いを一般トランザクションと同様にすることによって、その性能インパクトを小さくすることが可能である。

【０１９３】そこで、巨大トランザクションが一般トランザクションと変わらない程度しかメタデータを更新しないと分かった時点で、この巨大トランザクションの扱いは一般トランザクションと同じにする（デグレード）。

【０１９４】巨大トランザクションはその開始時には一般ログバッファの１つを一般トランザクション開始時と同様に与えられる。しかし、１つの巨大トランザクションが動作中は次の巨大トランザクションにログバッファを与えない点が一般トランザクションの場合と異なる。

【０１９５】巨大トランザクションがある程度以上のメタデータを更新することが分かった時点で、これまでの一般ログバッファの内容を巨大ログバッファへコピーし、そこで処理を継続する。

【０１９６】しかし、それほど多くのメタデータを更新しないと判断されれば、そのまま一般ログバッファで処理が継続され、また、次の巨大トランザクションに対し、処理の開始（ログバッファに使用）を許可する。

【０１９７】こうして、巨大トランザクションを途中から一般トランザクションとして扱うこと（デグレード）を実現する。

（２．２．１）デグレード契機・サブトランザクション化契機

巨大トランザクションは複数のサブトランザクションに分割される場合もあれば、一般トランザクションへデグレードする場合もある。それぞれの判定基準は以下の通りである。

◎デグレード判定基準
・一般ログバッファの残りサイズで、将来全てのメタデータ更新が完了できると判断できた場合。
◎巨大トランザクション用ログバッファへの移行契機
・一般ログバッファの残りサイズが次のステップの最大変更量よりも少ない場合。
◎サブトランザクション判定基準
・巨大ログバッファの残りサイズが次ステップの最大変更量よりも少ない場合。

（２．３）一般トランザクションのログ採取

一般トランザクションは、以下の手順に従いログを採取する。

１）ＬＯＧ＿ＢＥＧＩＮ
ａ）利用するログバッファを確定する。
【０１９８】ｂ）ログバッファの先頭にＢＥＧＩＮマークを作成する。
２）各メタデータの更新
ｃ）更新されたメタデータがリリースされる（ロックが解放される）直前に、更新されたメタデータについて、ヘッダ（更新情報）をログバッファに作成し、更新内容をログバッファにコピーする。
【０１９９】ｄ）更新されたメタデータをリストに繋ぐ。
３）ＬＯＧ＿ＥＮＤ
ｅ）ＥＮＤマークをログバッファに作成する。
【０２００】ｆ）ログバッファをログライトバッファへコピーする。

ｇ）ログライトデーモン１０５がログボリューム１２０へ反映する。
ｈ）このトランザクションで立てられた追い出し不可フラグを落とす。
（２．３．１）ＬＯＧ＿ＢＥＧＩＮ
トランザクション開始を意味する。同一関数内にＬＯＧ＿ＥＮＤの宣言が必要。並行動作可能数だけ既にトランザクションが動作していたら、それらのどれか１つが終了するまで寝て待つ。
【０２０１】ａ）現在実行中のトランザクション数がログバッファの数と等しい場合、それは既に許された並行動作数だけトランザクションが実行中であることを意味する。そのため、他トランザクションが終了するまで寝る。そのような状態でなければ、ログバッファの利用状況を管理するビットマップより未利用状態のログバッファを１つ選び出し、そのビットを立てる。さらに、並行動作数カウンタをインクリメントする。
【０２０２】ｂ）前述のＢＥＧＩＮマークをログバッファの先頭に作成する。
（２．３．２）各メタデータの更新
メタデータを参照した場合にはログ採取の必要はない。更新した場合のみログを採らなければならない。
【０２０３】ｃ）メタキャッシュ１３０に読み込んだメタデータについて処理が終了し、メタキャッシュ１３０より解放するための関数（リリース関数）を呼び出す際、「更新した」ことを明示された場合、ログを採取する。
【０２０４】ｉノードについてはログ対象トランザクション内でのロック解放時がログ採取契機でおる。具体的には、上記で使用権を得たログバッファにヘッダを作成し、メタデータの更新内容をコピーする。このとき、メタデータの種類によってその手法が若干異なる。
【０２０５】ｃ－１）ビットマップの場合：ヘッダには獲得（解放）したメタデータ番号を入れ、更新内容は「０→１」「１→０」とビット操作だけとする。
ｃ－２）メタデータ本体の場合：更新内容にはメタデータをそのままコピーする。
【０２０６】ｃ－３）スーパブロックの場合：巨大トランザクションがデグレードした場合のみ、一般ログ内でのスーパブロックのログ採取がありうる。その時刻（シーケンシャル番号）とデータ空きサイズを記録する。
【０２０７】コピーするとともに、ログバッファのターミナルで管理されるログサイズに、っこで作成したログのサイズを加える。
ｄ）メタキャッシュ１３０の大きさは有限であることから、トランザクションが進行するためには、必要のないメタデータをメタキャッシュ１３０から追い出し、その場所に必要なメタデータを読み込むという処理を行う追い出し機構がある。更新されたメタデータはログを書き出すまでは追い出されてはならない。そのために、このメタデータをメタライトリストと呼ぶ、メタボリュームへの反映を司るメタライトデーモン１０４が参照するリストへ繋ぐ。
（２．３．３）ＬＯＧ＿ＥＮＤ
トランザクションの終了を示す。ここで、ＥＮＤマークの作成を行う。ログボリューム１２０反映はログライトデーモン１０５による作業である。
【０２０８】ｅ）ＬＯＧ＿ＢＥＧＩＮにて作成したＢＥＧＩＮマークに対応するＥＮＤマークをログバッファの末尾に作成する。
ｆ）利用したログバッファの内容を追加モードにあるログライトバッファにコピーする。この時にログ番号を決定し、ＢＥＧＩＮマーク、ＥＮＤマークに埋め込むログバッファのターミナルに記録していたログのサイズをログライトバッファのサイズに加える。同期書き込み要求の場合は、ログライトデーモン１０５（詳細後述）が正常にログボリューム１２０に反映したことを待つ必要があるが、同期書き込み要求でない場合には、ログバッファを解放（ビットを落とす）して終了する。
【０２０９】ｇ）後述するログライトデーモン１０５が、ログライトバッファの内容をログボリュームへ反映する。
ｈ）リリース時に立てた追い出し不可フラグを全て落とす。ただし、フラグを落とすのはログライドデーモンによってなされるため、このトランザクションはそのまま終了する。
（２．４）巨大トランザクションのログ採取
巨大トランザクションも、一般トランザクションとほぼ同様の処理だが、最大の違いは、トランザクションの状況によってログバッファをサイズの大きい巨大ログバッファへ移行したり、サブトランザクションとして分割したログ採取を行わなければならない場合があることである。
１）ＬＯＧ＿ＢＥＧＩＮ
ａ）利用するログバッファを確定する。
【０２１０】ｂ）ログバッファの先頭にＢＥＧＩＮマークを作成する。
ｃ）ＢＥＧＩＮマーク作成と同時に、オペレーションログをログバッファに作成する。
２）各メタデータの更新
メタデータがリリースされた時に「更新したこと」が明示されたら、
ｄ）更新メタデータが空き管理情報の場合、更新メタデータがリリースされる度にＢＴＦリストあるいはＢＴＡリストとよぶ空き管理メタデータのために用意するリストに繋ぐ。この時、既にリストに繋がれていればリスト構造には手を加えない。
【０２１１】ｅ）更新メタデータが空き管理情報以外の場合、更新されたメタデータがリリースされる度にヘッダをログバッファに作成し、更新内容をログバッファ域

にコピーする。

【０２１２】ｆ）更新メタデータをリストに繋ぐ。まだ一般ログバッファで処理している場合、

ｇ）一般ログバッファで足りるか判定する

ｇ－ａ）足りないなら巨大ログバッファへこれまでの内容をコピー。

【０２１３】ｇ－ｂ）足りると確定でき、かつ、デグレード条件を満たせば、次の巨大トランザクションを受け付ける。

ｇ－ｃ）まだ判断ができなければそのまま継続する。既に巨大ログバッファで処理している場合、

ｈ）サブトランザクション化するか判定する

ｈ－ａ）するなら、ＢＴＦリスト及びＢＴＡリストをたどりながらそれらをログバッファにコピー、ＥＮＤマークを作成し、ログライトバッファへコピーする。ログライトデーモン１０５を起動し、ログを出力する。

【０２１４】ｈ－ｂ）まだしないなら、そのまま継続する。

３）ＬＯＧ＿ＥＮＤ

ｉ）最終ＥＮＤマークをログバッファに作成する。

【０２１５】ｊ）ログバッファをログライトバッファへコピーする。

ｋ）ログライトデーモン１０５がログボリューム１２０へ反映する。

ｌ）このトランザクションで立てられた追い出し不可フラグを落とす。

（２．４．１）ＬＯＧ＿ＢＥＧＩＮ

巨大トランザクションも最初は一般ログバッファを用いる。

【０２１６】ａ）一般トランザクションの場合は、現在の並行動作トランザクション数とログバッファの数との関係によってログバッファを獲得できるかどうかが定まった。巨大トランザクションの場合は、現在他の巨大トランザクションが動作中か否かが問題であり、一般トランザクションの並行動作数とは関係しない。巨大トランザクションが動作中か否かのフラグを調査し、非動作中であれば、そのフラグを立て、ログバッファの利用状況を管理するビットマップより未利用状態のログバッファを１つ選び出し、そのビットを立てる。巨大トランザクションが現在動作中であれば、終了まで寝て待つ。

【０２１７】ｂ）ＢＥＧＩＮマークを作成する。ここで、トランザクションタイプに巨大トランザクションとなりうるトランザクションが指定されていた場合には、必ず後ろにはオペレーションログが付随する。対応するＥＮＤマークがないログはリプレイしてはならない。また、サブトランザクション化しているのに、最終ＥＮＤマークがないなら、オペレーションログのリプレイが必要である。

【０２１８】ｃ）巨大トランザクションをサブトランザクションログとして分割する場合にはオペレーションログを採取する必要がある。ここで、オペレーションログとは、各ログ採取対象関数に与えられたパラメタが、将来リプレイすることが可能な形に変更されたものである。具体的には、メモリアドレスで与えられるパラメタは、メタボリューム内のオフセットに変更して記録する。

（２．４．２）各メタデータの更新

ｄ）更新メタデータが空き管理情報である場合、一回のトランザクションで同一の領域が何度も変更される場合が考えられる。その都度、ログを採取しているとインコアログやログボリュームがフルになる可能性が高まる。これを回避するために、更新情報をリストによって管理し、複数回更新された空き管理メタデータを１つにまとめてログに記録する。具体的には、メタデータの状態毎にリスト管理する専用のＢＴＦリスト及びＢＴＡリストに繋がれる。このリストに繋がれていれば、そのデータはダーティであることを意味する。初めて更新するデータであれば、ターミナルから繋がるリストに追加する。既にリストに繋がれているのであれば、２回目以降の更新であることを意味し、ポインタについてはそのままで良い。リストにメタデータを追加する場合には、ログサイズを更新する。

【０２１９】ｅ）更新メタデータが空き管理情報以外であれば、同一メタデータに対して何度もホールド／リリースが発行されるとは考えられないので、リリースの度（ロック解放の度）にログバッファへコピーする（一般トランザクションの場合と同じ）。ログサイズを更新する。

【０２２０】ｆ）既に何度か述べたが、更新されたメタデータがログよりも先にメタボリューム１１１～１１３へ反映されてはならない。この追い出し不可状態もリスト構造によって管理される。

【０２２１】ｇ）この巨大トランザクションが更新するメタデータが、以降、どれだけの数のメタデータを更新するかを算出することによって、処理が分かれる。

ｇ－ａ）現在利用している一般ログバッファの残りでは次ステップの実行によるメタデータの更新の全てをまかないきれない場合があると判断した場合には、巨大ログバッファへの移管を行う。

【０２２２】ｇａ－１）これまでに溜まったログバッファの内容を巨大ログバッファへコピーする。この時、ＢＴＦリストはそのまま繋いだままとし、ＢＴＡリストは巨大トランザクション用のターミナルへ移動する。

【０２２３】ｇａ－２）巨大ログバッファを利用して、その巨大トランザクションは処理を継続する。

ｇ－ｂ）現在利用している一般ログバッファの残りだけで、以降のメタデータ更新を全て記録できると判断できるならば、この巨大トランザクションは一般トランザクションにデグレードする。ＢＴＦリストにメタデータが繋がれている場合はデグレード条件を満たさないため、

ここでの処理はありえない。ＢＴＦリストはそのまま利用を続ける。

【０２２４】ｇｂ－１）巨大トランザクションが動作中か否かを示すフラグを落とす。

ｇｂ－２）一般トランザクションの並行動作数カウンタをインクリメント。

ｇｂ－３）次の巨大トランザクションが寝ているならば起こして、実行を受け付ける。

【０２２５】ｇｂ－４）このまま一般ログバッファを利用して処理を継続する。

ｇ－ｃ）まだ判断できないのであれば、このまま一般ログバッファを利用して処理を継続する。

【０２２６】ｈ）サブトランザクション判定基準（前述）に基づき、このトランザクションをサブトランザクション化するかどうか判定する。

ｈ－ａ）サブトランザクションに分割する場合は、

ｈａ－１）更新された空き管理情報を、リストを辿りながらログバッファへコピーする。

【０２２７】ｈａ－２）ＥＮＤマークを作成する。

ｈａ－３）ログライトバッファへコピーする。

ｈａ－４）ログライトサイズを更新する。

【０２２８】ｈａ－５）ログライトデーモン１０５を強制起動する。

ｈａ－６）ここで変更した全てのメタデータの追い出し不可フラグを落としてメタライトリストに繋ぐ。これにより，メタライトデーモン１０４は任意の時ににこれらのメタデータを書き出すことが可能となる。

【０２２９】ｈａ－７）巨大ログバッファの先頭にＢＥＧＩＮマークを作成する。

ｈａ－８）オペレーションログを作成する。

ｈ－ｂ）サブトランザクションに分割する必要がない間は、巨大ログバッファ内でそのまま継続する。

（２．４．３）ＬＯＧ＿ＥＮＤ

トランザクションの終了を示す。ここで、ＥＮＤマークの作成を行う。実際のログボリューム１２０反映はログライトデーモン１０５の仕事である。

【０２３０】ｉ）最終ＥＮＤマークを作成する。最終ＥＮＤマークは通常のＥＮＤマークとマジックワードが異なるだけである。

ｊ）利用したログバッファの内容を追加モードにあるログライトバッファへコピーする。この時にログ番号を決定し、ログサイズをログライトサイズに加える。同期書き込み要求の場合は、ログライトデーモン１０５が正常にログボリューム１２０反映したことを待つ必要があるが、その他の場合には待たないで次の処理へ進む（すなわち非同期書き出しである）。巨大トランザクション動作中フラグを落とし、寝て待つ次の巨大トランザクションを起こす。

【０２３１】ｋ）ログライトデーモン１０５が、ログライトバッファの内容をログボリュームへ反映する。

ｌ）更新したメタデータをログバッファへコピーした時に立てた追い出し不可フラグを全て落とす。ただし、フラグを落とすのはログライトデーモン１０５によってなされるため、このトランザクションはそのまま終了する。

（２．５）ログライトデーモン

並行して動作し、順次終了するトランザクションによって更新されたメタデータのログは専用の書き出しデーモン（ログライトデーモン１０５）によってログボリューム１２０へ反映する。このデーモンはマウント時に起動され、アンマウント時に停止する。すなわち、ファイルシステム毎にスレッドを生成する。

【０２３２】各トランザクションが終了すると、ログバッファ内にＥＮＤマークが作成され、このログバッファの内容はログライトバッファ１５０にコピーされる。このログライトバッファ１５０は複数のログバッファの内容を一括してログボリューム１２０に反映するためのものであり、そこにはメモリコピーの負担があるが、何度もＩ／Ｏを発行するよりは良いと考える。

【０２３３】ログライトバッファ１５０へコピーされると、そのログバッファは利用中バッファリストから空きバッファリストへと繋ぎ直され、かつ、トランザクションが同期書き込み要求でなければ、動作中トランザクション数をデクリメントする。これにより、次トランザクションの実行が進むようになる。

【０２３４】ログライトデーモン１０５はログライトバッファの内容を、定期的、あるいは同期書き込み指定のトランザクションがある場合などにログボリュームに反映する。反映している間（Ｉ／Ｏ中）は２本あるログライトバッファのうち、もう片方のログライトバッファに内容をコピーしてそのトランザクションは処理を終了する。

（２．５．１）処理手順

ログライトデーモン１０５単体の処理手順はそれほど複雑ではない。

【０２３５】ａ）現在のログライトバッファをライトモードにし、もう片方を追加モードにする。

ｂ）ログライトバッファの内容をログボリューム１２０へ同期書き出しする。

【０２３６】ｏ）場合に応じて有効範囲情報を書き出す。

ｄ）ログ待ちリストに繋がるメタデータを、メタライトリストへ繋ぎかえる。

ｅ）規定時間の間、スリープする。

【０２３７】ｆ）規定時間が経過、または他の要因で起こされたらａ）へ。

以下、詳細説明である。

ａ）Ｉ／Ｏ中はその対象域に対して追加更新は許されない。そのため、現在のログライトバッファが書き出しが終わり、次の書き出しが始まるまでは、もう片方のログ

ライトバッファへ終了したログバッファをコピーするよう設定する。Ｉ／Ｏ中のログライトバッファをライトモード、ログバッファの内容をコピーする方を追加モードと呼ぶ。切り替えはこのデーモンによって行われ、各トランザクションは常に追加モードにあるログライトバッファに自ログバッファの内容をコピーする。

【０２３８】ｂ）ログライトバッファに新しい内容が含まれていないのであれば、Ｉ／Ｏを発行する必要はない。そこで、まずログライトサイズを調べ、ゼロであればＩ／Ｏを発行せず、タイマを指定して再び寝る。書き出すべきログが存在するのであれば、以下の手順に従う。

【０２３９】ｂ－ａ）ログボリュームの空きサイズ判定。

ログボリュームの先頭オフセット（Ａ）、メタライトリスト先頭のメタデータを管理する構造の上書き可能オフセット（Ｂ）を調べる。それとログライトオフセット（Ｃ）、及びログボリュームの最終オフセット（Ｄ）から以下の手順となる。

・Ａ＜Ｂ≦Ｃ＜Ｄ

①ログライトサイズが（Ｄ－Ｃ）以下ならば、Ｃから書き出す。

②ログライトサイズが（Ｄ－Ｃ）より大きく、かつ、（Ｂ－Ａ）以下ならば、Ａから書き出す。

・Ａ≦Ｃ＜Ｂ≦Ｄ

③ログライトサイズが（Ｂ－Ｃ）より小さければ、Ｃから書き出す。

【０２４０】これらの条件に合致せず、ログの出力ができない場合には、メタライトデーモン１０４を起動して、自分はスリープする。メタライトデーモン１０４の動作により起動されたら、再度、空きサイズ判定から行う。

【０２４１】ｂ－ｂ）ログボリューム１２０へ同期書き出しする。

ｂ－ｃ）エラー判定。同期書き出し後、エラー判定を行う。エラーが発見された場合の処置については後述（２．５．３）する。

【０２４２】ｂ－ｄ）領域判定。

書き出しが終わった後、再び、ｂ－ａ）と同様な空き領域の判定を行う。ここで、残りが少ないと判定された時、その残りの大きさに応じてメタライトデーモン１０４を「平常起動」「緊急起動」する。

【０２４３】ｃ）ファイルシステム復元に必要なログは、メタライトリスト先頭の上書き可能オフセットからたった今書き出した位置までである。そこで、それを有効範囲情報としてディスクへ書き出す。ここで、書き出しはログのシーケンシャル性を考慮し、ある程度の間隔をおいて行う。ここでは、回数に応じて、数回に一回、書き出すこととした。

【０２４４】ｄ）ｂ）にて書き出したログに含まれるメタデータは全て「ログ未反映状態」、すなわち追い出し不可状態としてリスト（ログ未反映リスト）に繋がれているはずである。そこで、ログ未反映リストを辿りながら、そこに繋がるメタデータをメタライトリストに追加することにより、メタライトデーモン１０４が任意の時にこれらのメタデータをメタボリュームに反映できるようになる。

【０２４５】ｅ）ログもシステム全体から見れば非同期書き出しとし、トランザクションが同期書き込み要求でなければ、ログを書き出す前にそのトランザクションは終了することができる。さらにＩ／Ｏ発行回数を削減するために、複数のトランザクションを一括して書き出すために、しばらくの間スリープし、その間にログライトバッファへ複数トランザクションのログを溜める。

【０２４６】ｆ）規定時間後には自分で起きて、処理を繰り返す。しかし、他要因によって突然起こされて処理を始める場合もある。「その他の要因については後述（２．５．２）する。

（２．５．２）動作契機

ログライトデーモン１０５は以下の契機でログライトバッファの内容をログボリュームへ書き出す。

◎定周期

ログライトデーモン１０５は一定周期毎にスリープ状態から自発的に目覚め、ログライトバッファ１５０の内容をログボリューム１２０へ反映する。

◎同期書き込み要求のトランザクション

トランザクションによっては同期書き込み要求で呼ばれる場合がある。このトランザクションについては、ログの書き出しを待たなければ終了してはならない。そのため、ＬＯＧ＿ＥＮＤ呼び出し後に、明示的にＬＯＧ＿ＳＹＮＣを行う。ＬＯＧ＿ＳＹＮＣはログライトデーモン１０５が寝ている場合には起こし、ログライトバッファの内容をその場で書き出させる。

【０２４７】現在ログライトデーモン１０５がＩ／Ｏ中であったら、ログライトデーモン１０５の処理が終わるのを寝て待つ。

◎ログライトバッファ不足定

周期毎にログライトバッファをクリアしても、大きなログバッファが連続して終了すると、その前にログライトバッファがフルに近い状態になることがありえる。この時にはログライトデーモン１０５が起こされ、ログライトバッファ１５０の内容をログボリューム１２０に書き出してクリアする。

【０２４８】具体的には、あるトランザクションが自ログバッファの内容をＬＯＧ＿ＥＮＤによってコピーしようとした時、ログライトバッファ１５０の残りサイズが自ログよりも小さいと判断された時、ログライトデーモン１０５を起こし、追加モードのログライトバッファ１５０を切り替えてもらい、その間は寝て待つ。

【０２４９】現在Ｉ／Ｏ中であり、かつ追加モードのロ

グライトバッファ１５０が足りなくなった時には、そのトランザクションは待ち合わさざるを得ない。

◎メタキャッシュ不足

トランザクション実行中に、メタキャッシュ１３０域のいずれかのメタデータを追い出さなければ必要なメタデータをメタボリューム１１１～１１３から読み込むことができずに処理が進まなくなる場合が考えられる。

【０２５０】そのため、トランザクションが現在キャッシュ上のメタデータを追い出そうとする時、メタキャッシュ１３０上のメタデータが全てダーティであり、かつ、ログ待ちリストに繋がれたメタデータが存在する場合には、ログライトデーモン１０５を起動する。

◎アンマウント時

アンマウント時には必ず書き出さなければならない。そのため、アンマウント処理の延長でログライトデーモン１０５を起こし、書き出しを行う。この時、アンマウントを実行するため、該当ファイルシステムにメタデータを更新するようなトランザクションは動作していないことが保証される。従って、現在のログライトバッファの内容を書き出せば、それで全てである。

【０２５１】ログリアとバッファの内容を書き出した後、ログライトデーモン１０５は終了する。

（３）メタデータ管理

次に、メタデータの管理方式について説明する。メタキャッシュ１３０内のメタデータはｍｅｔａｌｉｓｔ構造体によって管理される。ｍｅｔａｌｉｓｔ構造体には以下のメンバがある。

・メタデータへのポインタ
・ＴＲＡＮＳ（トランザクション）リストポインタ
・メタライトリストｐｒｅｖポインタ
・メタライトリストｎｅｘｔポインタ
・ログ待ちリストｐｒｅｖポインタ
・ログ待ちリストｎｅｘｔポインタ
・ログシーケンス番号
・ログボリューム内オフセット
・トランザクション番号
・状態フラグ
・メタデータタイプ
・ｂｕｆ構造体実体

（３．１）メタデータの状態遷移

ｍｅｔａｌｉｓｔ構造体には次の６種類の状態があり、４種類のリストに繋がれる。それはｍｅｔａｌｉｓｔ構造体のフラグによって示され、それぞれターミナルを別とするリストに繋げられることによって実現する。全てのｍｅｔａｌｉｓｔ構造体はいずれかのリストに繋がっており、例外はない。

Ａ）空き管理構造更新状態

データ空き領域を管理するメタデータがトランザクションによって更新されると、リストに繋ぎ、将来まとめてログバッファにコピーすることは既に述べた。トランザクション終了時に本リストに繋がれたメタデータが直接ログボリュームへ反映される。リストは並行動作数に応じて必要である。このリストをＢＴＦリストと呼ぶ。

【０２５２】この状態にあるメタデータはまだログバッファへはコピーされておらず、同一トランザクションによって再度更新される場合がある。既にリストに繋がれているメタデータであった場合は、リスト状態は変更しない。当然、メタボリュームに反映してはならない。

【０２５３】ＢＴＦリストは単方向環状リストであり、トランザクション途中状態と同じメンバを用いて繋がれている。

Ｂ）利用域管理構造更新状態

実施例ではファイルシステムをエクステント管理している。ｉノードから導かれる、そのファイルが利用しているエクステントを示す、間接エクステントブロックについても、トランザクション内で１つの間接エクステントブロックが何度も更新される場合が考えられる。そこで、空き管理構造の場合と同様に、トランザクション中は独自のリストに繋ぎ、トランザクション終了時にまとめてログバッファへコピーする。リストは並行動作数に応じて必要である。このリストをＢＴＡリストと呼ぶ。

【０２５４】この状態にある間接エクステントブロックはまだログバッファへはコピーされておらず、同一トランザクションによって再度更新される場合がある。既にリストに繋がれている間接エクステントブロックであった場合は、リスト状態は変更しない。当然、メタボリュームに反映してはならない。

【０２５５】ＢＴＡリストは単方向環状リストであり、トランザクション途中状態と同じメンバを用いて繋がれている。

Ｃ）トランザクション途中状態

トランザクション途中を示す。この状態のとき、該当するメタデータを更新したトランザクションは、ある１つのログバッファを専有しており、既にそこに更新後状態がコピーされている。リストは並行動作数に応じて必要である。このリストをＴＲＡＮＳリストと呼ぶ。

【０２５６】しかし、まだトランザクションが終了していないことからログがログボリューム１２０へ反映されておらず、本メタデータもメタボリュームへの反映はできない。この状態にあるメタデータはファイルのロックによって保護されているため、他トランザクションから参照・更新されることはない。

【０２５７】ＴＲＡＮＳリストのターミナルは配列になっており、その要素番号はトランザクションが用いているログバッファの番号（トランザクション番号）に対応する。

【０２５８】メタライトリストやログ待ちリストに繋がれているメタデータが繋がる場合がある。ＴＲＡＮＳリストは単方向環状リストであり、ＢＴＦリストが用いるメンバを併用する。

【０２５９】Ｄ）ログ待ち状態
トランザクションは既に終了しているが、ログバッファの内容はログライトデーモン１０５によってログボリューム１２０へ反映されるため、この時点ではまだログボリューム１２０に反映されていない状態である。（デーモンは同期ライトを行うが、トランザクションの視点から見れば、ログ反映が非同期に行われているように見える。）
トランザクションが終了する際に、Ｃ）の状態にあるＴＲＡＮＳリスト（巨大トランザクションの場合はＢＴＦリストも）をログ待ちリストに繋ぎかえる。このリストはトランザクションが終了する毎に後方へ伸びるリストであり、ログライトバッファと同数の２本存在する。
【０２６０】この状態にあるメタデータは既にトランザクションが終了しており、ログはログライトバッファにコピーされている。トランザクションが終了していることから、他トランザクションによって参照・更新される可能性がある。この時（他トランザクションによって状態が変化した時）、リスト位置は変更せず、状態フラグだけを変更する。
【０２６１】メタライトリストに繋がれたメタデータが、他トランザクションによって再度更新されたために、トランザクション途中状態になり、そのトランザクションが終了したことによって、ログ待ち状態になった場合には、このメタデータはメタライトリストにも繋がれている。
【０２６２】ログライトデーモン１０５によって、ログの反映が進むと、それに応じたメタデータをメタライトリストへ追加していく。この時、既にメタライトリストに繋がれているメタデータは、そのままの位置でいなければならない。
Ｅ）書き出し可能状態
これは、ログバッファのログボリュームへ反映が終了し、自由にメタデータをメタボリュームへ反映することができるようになった状態である。この状態にあるメタデータは他トランザクションによって参照・更新される可能性がある。この時、リスト位置は変更せず、状態フラグだけを変更する。
【０２６３】繋がるリストはメタライトリストであり、１本だけ存在する。メタライトデーモン１０４はこのリストを参照してメタボリュームへの反映を行う。
Ｆ）Ｉ／Ｏ中状態
この状態にあるメタデータは現在非同期ライトによってメタボリュームに対してＩ／Ｏを投げているが、まだ完了していない。Ｉ／Ｏ途中であるため、他トランザクションは操作することができない（参照することはできる）。メタライトリストにそのまま繋がれている。
（３－２）ログボリュームの上書き判定情報
ログボリュームはサイズが有限であるため、サイクリックに利用し、過去のログを上書きしてシステムは動作する。ここで、過去のログを上書きするためには、そのトランザクションが更新したメタデータが全てメタボリュームに反映されていることが必要である。上書き可能領域を管理するために、以下の構造を設け、メタライトデーモン１０４が変更、ログライトデーモン１０５が参照する。
◎ログシーケンス番号
各トランザクションが終了し、ログバッファの内容をログライトバッファへコピーする毎に与えられる、シーケンシャル番号である。ログボリューム内のＢＥＧＩＮマーク、ＥＮＤマークに含まれる番号と同一である。既にライトリストに繋がるメタデータが再度更新される場合にも、この番号は変更されない。
◎ログボリューム内オフセット
ログボリューム内にはトランザクション毎のログが連続して並んでおり、上書きするためには、それぞれのトランザクションが更新した全てのメタデータがメタボリュームへ反映されている必要がある。
【０２６４】同一のログ番号を持つｍｅｔａｌｉｓｔ構造体は、ここにはそのログの先頭オフセットを挿入する。ＬＯＧ＿ＥＮＤによってログバッファの内容がログライトバッファにコピーされる際、そのアドレスとログボリュームの書き出しオフセットから導かれる。トランザクション毎のログボリューム内オフセットがＴＲＡＮＳリストを辿りながらそれぞれのｍｅｔａｌｉｓｔ構造体に設定する。
【０２６５】ログライトデーモン１０５は、ログを書き出す際にメタライトリストの先頭に繋がるｍｅｔａｌｉｓｔ構造体を参照し、現在のポインタにログライトバッファに入っているログサイズを足した位置が、このオフセット値を超えないことを確認した後で書き出しを行う。
（３．３）メタボリュームへの反映
トランザクションによって更新されたメタデータは、トランザクションが終了しログがログボリューム１２０に反映されるまでは書き出されてはならない。そのために、メタキャッシュ１３０上の各メタデータはそれぞれの状態に応じてリストに繋がれ、唯一メタボリュームへの反映が許可されているリストはメタライトリストである。
【０２６６】時間のかかるＩ／Ｏ要求をトランザクション内で行うことを避けるために、メタデータのメタボリュームへの反映はトランザクションとは関係ないところで動作するデーモンに委ねる。このデーモンはメタライトリストだけを意識し、ｍｅｔａｌｉｓｔ構造体内の情報から状態に応じてＩ／Ｏ発行するかどうかを決定し、関連付けられたｂｕｆ構造体を用いてメタデータをメタボリュームへ反映する。
【０２６７】デーモンはメタライトリストに繋がれたメタデータを書き出した後、そのメタデータをリストから

外し、ｃｌｅａｎであると設定する。

（３．４）メタライトデーモン

メタライトデーモン１０４は、マウント時に起動され、アンマウント時に停止する。すなわち、ファイルシステム毎にスレッドを起動する。

【０２６８】ログ書き出しの終わったトランザクションが更新したメタデータは、それを管理するｍｅｔａｌｉｓｔ構造体が全てメタライトリストに繋がれている。メタライトデーモン１０４はメタライトリストに繋がるメタデータを、ログボリュームの残りが少なくなったと判断された場合などに非同期でメタボリュームに反映する。反映している間は該当するメタデータは更新することができず、Ｉ／Ｏ終了を待ち合わせることになる。

【０２６９】メタライトデーモン１０４はメタライトリストを意識し、繋がるメタデータをメタボリュームへ反映する。これにより、ログボリュームの上書き可能領域が拡大する。

【０２７０】しかし、メタライトリストに繋がれているメタデータの中にも、再度トランザクションによって更新が進み、メタボリュームへの反映が禁じられているものが存在する場合がある。この時、他のメタデータを書き出すとしても、そのメタデータについては非同期ライトの発行を待ちあわせなければならない。このようなメタデータが存在すると、以降のメタデータを全て書き出せたとしても、上書き可能領域を拡大することができない。

【０２７１】メタライトデーモン１０４は他者から起動されて処理を開始する場合と、自発的に起動する場合がある。以下に処理手順を述べるが、システムの状態（ログボリュームの空きやメタキャッシュ１３０の空きなど）に応じて動作が若干異なる。また、ここで述べる処理手順にはデーモン内だけではなく、ドライバから呼ばれる関数での処理も含まれている。

（３．４．１）自発起動処理

メタライトデーモン１０４はタイマによって自発的に起動して、それまでにメタライトシストに繋がれた書き出し可能なメタデータをメタボリュームに反映する。この時、メタライトリストの全てを書き出す訳ではなく、ある一定の数だけ非同期書き出しを行う。普段から少しずつメタデータの書き出しを行っておくことによって、資源不足による起動を避け、Ｉ／Ｏ負荷分散を図る目的がある。

【０２７２】ａ）自発起動時には、メタライトリストに繋がるメタデータ数を調査する。

ｂ）メタライトリストの先頭から、メタデータの状態を調べる。

ｃ）メタデータが書き出し可能状態であれば、その状態をＩ／Ｏ負荷態に変更し、非同期ライトを発行する。

【０２７３】ｄ）ｂ＿ｉｏｄｏｎｅの関数がｌ／Ｏの成功を確認する。

ｅ）ｂ＿ｉｏｄｏｎｅの関数がメタライトリストから外す。

ｆ）規定数だけ繰り返す

ｇ）スリープする。

【０２７４】以下、詳細説明である。

ａ）自発起動間隔として定義する間隔毎にメタライトリストに繋がるメタデータ数を確認する。具体的には、メタライトリストのターミナルに含まれる、リンクされたメタデータ数を調べる。このとき、それほど多くのメタデータが繋がれていない場合には書き出しは行わない。

【０２７５】ｂ）メタライトリストには基本的にはログの書き出しが終わった、書き出し可能状態のメタデータが繋がれている。しかし、トランザクションが終了してメタライトリストに繋がれた後で他トランザクションによって更新されると、そのメタデータだけは書き出し不可の状態に戻ってしまう。そのため、メタライトデーモン１０４はメタライトリストに繋がるメタデータの状態を確認しながら書き出し処理を行わなければならない。

【０２７６】ｃ）現在ポイントするメタデータが書き出し可能状態であれば、その状態をＩ／Ｏ状態に書き換え、ｍｅｔａｌｉｓｔ構造体に含まれるｂｕｆ構造体を用いてＩ／Ｏを発行する。Ｉ／Ｏ中状態のメタデータは他トランザクションから更新されることはない。メタデータのディスクライトは非同期ライトで行い、デーモンはＩ／Ｏの結果を待たずに次の処理へ進む。

【０２７７】メタデータが書き出し不可状態であるなら、そのメタデータは諦め、リストの次に繋がるメタデータに進み、状態を調べる。

ｄ）非同期ライトによって発行されたＩ／Ｏの結果を判定するのは、ｂｕｆ構造体のメンバｂ＿ｉｏｄｏｎｅに組み込まれた関数である。この関数にはｂｕｆ構造体が引数に与えられ、それを元にエラー判定処理を呼び出す。ここでエラーを検出した場合には、異常系処理へ進む（後述）。成功していた場合には、該当メタデータのｃｌｅａｎ数をインクリメントする。

【０２７８】ｅ）ｂ＿ｉｏｄｏｎｅに組み込まれた関数はメタライトリストの管理も行う。引数に渡されたｂｕｆ構造体の上を見るとそこはｍｅｔａｌｉｓｔ構造体になっており、そのフフグからＩ／Ｏ中フラグを落とし、メタライトリストから外す。また、メタデータをダーティ状態からｃｌｅａｎ状態へ変更する。これは、メタキャッシュ１３０で管理されるメタデータである場合には、それぞれの管理構造体で管理されており、この管理構造体はｍｅｔａｌｉｓｔ構造体からポイントされる。

【０２７９】全ての処理が終了したら、このｍｅｔａｌｉｓｔ構造体をリリースする。

ｆ）メタライトリストに繋がる書き出し可能メタデータを、次々とリストを辿りながら定義した数だけ処理を繰り返す。ここで、書き出しが不可とされているメタデータについては、この数には含めない。また、メタライト

リストが定義数よりも少ない場合には、当然そこで終了となる。

ｇ）スリープする。再度、自発的に起動するか、あるいは資源不足によって他から起動されるまで、スリープは継続する。

（３．４．２）平常処理

システム状態が以下の場合には、ここで述べる処理手順に従いメタライトデーモン１０４は動作する。

◎ログボリュームの残りが少ない。

【０２８０】ログライトデーモン１０５が判断する。ログライトバッファを書き出す際に、上書き可能域の大きさを計算し、この閾値を下回った時にメタライトデーモン１０４を起動する。

◎メタキャッシュの残りが少ない。

【０２８１】更新リリースがあった時、各メタデータのｃｌｅａｎ数がデクリメントされるが、その数がこの閾値を下回った時にメタライトデーモン１０４を起動する。

ａ）メタライトリストの先頭から、メタデータの状態を調べる。

【０２８２】ｂ）メタデータが書き出し可能状態であれば、その状態を１／Ｏ中状態に変更し、非同期ライトを発行する。

ｃ）ｂ＿ｉｏｄｏｎｅの関数がＩ／Ｏの成功を確認する。

【０２８３】ｄ）ｂ＿ｉｏｄｏｎｅの関数がメタライトリストから外す。

ｅ）メタライトリストの最後まで繰り返す。

ｆ）スリープする。

【０２８４】以下、詳細説明である。

ａ）～ｄ）自発的に起動した場合と同じである。

ｅ）メタライトリストを辿りながら、繋がる全ての書き出し可能メタデータについて処理を繰り返す。平常処理時には、書き出し不可のメタデータについては飛ばして処理を行う。そのため、ログボリューム不足時には、メタライトリストの先頭、すなわち、ログボリュームの上書き可能位置を制限しているメタデータが書き出せなければ資源不足は解消しないが、平常処理であるため、ここでは特別な対処を行わない。

【０２８５】ｆ）タイマを設定してスリープする。

（３．４．３）緊急処理

システム状態が以下の場合には、ここで述べる処理手順に従いメタライトデーモン１０４は動作する。

◎ログボリュームの残りが非常に少ない。

◎メタキャッシュ１３０の残りが非常に少ない。

【０２８６】ａ）トランザクションの新規開始を制限する。

ｂ）メタライトリストの先頭から、メタデータの状態を調べる。

ｃ）メタデータが書き出し可能状態であれば、その状態

をＩ／Ｏ中状態に変更し、非同期ライトを発行する。

【０２８７】ｄ）ログ未反映状態となっているメタデータがあれば、ログライトデーモン１０５を起動する。

ｅ）ｂ＿ｉｏｄｏｎｅの関数がＩ／Ｏの成功を確認する。

【０２８８】ｆ）ｂ＿ｉｏｄｏｎｅの関数がメタライトリストから外す。

ｇ）繰り返す。

ｈ）現在の資源状態を調査し、不足があれば、それが解消するまで繰り返す。

【０２８９】ｉ）トランザクションの受付を開始する。

ｊ）スリープする。

以下、詳細説明である。

【０２９０】ａ）緊急時には、新規のトランザクションの開始を受け付けない。具体的には、トランザクションに利用するログバッファを与えないことによって、そのトランザクションのＢＥＧＩＮ宣言時にてスリープさせる。そのために、特定のフラグを設け、ＢＥＧＩＮ宣言時にそのフラグを参照しなければならない。

【０２９１】ｂ）～ｇ）ここは基本的に平常処理の場合と同じ処理である。すなわち、メタライトリストの先頭から、追い出し不可状態のメタデータは飛ばして、書き出し可能状態のメタデータを順に非同期ライトによって書き出す。

【０２９２】ただし、ｄ）だけ異なる。

ｄ）ログボリュームへの書き出しがデーモンによって行われることから、たとえトランザクションが終了していてもログが未反映のためにメタボリュームへの反映を拒否しているメタデータが存在することが考えられる。この場合は、強制的にログライトデーモン１０５を起動して、書き出し可能状態へ移行するように操作する。既にログライトデーモン１０５が動作中であれば、そのまま先へ進む。

【０２９３】ｈ）ここで、現在の資源状態を調べる。平常処理の動作契機となる閾値以上の資源が回復していなければ、再度、メタライトリスト内のメタデータ書き出しを試みる。ここで、ｄ）によってログ未反映状態のメタデータが、書き出し可能となったことによって進展することを期待している。複数回、繰り返すことによって、新規トランザクションの受付を制限していることから、資源回復までメタデータの書き出しが可能であると考える。

【０２９４】ｉ）ａ）にて制限していたトランザクションの受付を再開する。

ｊ）タイマを設定して、スリープする。

（４）獲得解放処理

メタデータの割り当て状況を管理するビットマップの状態として、FREE-dirtyとALLOC-dirtyを区分し、FREE-dirtyなビットマップからは獲得しないようにする。

【０２９５】具体的には以下の通りである。

・マウント時にビットマップを読み込む際に、１つを獲得用と定める。簡単のため、複数読み込むビットマップのうち、一番若いもの（最初に読み込むもの）を最初の獲得用ビットマップとする。
・獲得用ビットマップの複製を作成する。
・獲得時には複製を検索して獲得位置を決定し、本体と複製の両方のビットを操作する。
・解放時には本体のみのビットを操作する。
・複製ビットマップには解放が記録されないため、獲得処理が進むと全てのビットが立ち、そのビットマップでは獲得できないと判断される。その場合、現在メモリ上にあるビットマップのうち、ＣＬＥＡＮなもの、またはALLOC-dirtyのビットマップを次の獲得用ビットマップと定義し、その複製を作成する。
・メモリ上のビットマップが全てFREE-dirtyである場合には、どれかを追い出し、新しいビットマップを獲得用ビットマップとして読み込む。そして、その複製を作成する。ここで、追い出し・読み込みのアルゴリズムは従来通りで良い。

【０２９６】獲得用ビットマップとして選ばれたビットマップは追い出しの対象とはしない。そのためメタキャッシュ域不足によって他から追い出されることはない。また、メタライトリストに繋がったことによって、メタボリュームに反映された場合にも、ＣＬＥＡＮな状態にはなるが、複製は更新せず、そのままとする。

【０２９７】本実施の形態による効果は、以下の通りである。以上説明したように、本発明によれば複数の二次記憶装置に保存されたメタデータのログにボリューム情報を含め、また、有効なログの位置を算出・保存することでリプレイするログ量を減少せしめ、さらに、ログボリューム全体のゼロクリアをする必要をなくすことにより、ログ機構の最大の利点であるところのファイルシステム復元時間の短縮に及ぼす影響を小さくし、オーバオールコンピュータシステム可用性の増大に寄与するところが大きい。

【０２９８】さらに、本発明によれば同一トランザクション内で複数回更新されるメタデータのログを一度しか採取せず、また、ログバッファの分割や、獲得・解放処理の特殊なログ採取方式によって複数のトランザクションの独立性を考慮し、かつ、ログ機構導入による速度性能の劣化を最小に留めることにおいて寄与するところが大きい。

【０２９９】加えて、本発明によればトランザクション毎に異なるログの採取量を考慮し、多くのメタデータを更新するトランザクションについては分割し、トランザクションに与えられたパラメタをもログに採取し、リプレイ時にそのパラメタを元に、中途で終わったトランザクションを再度実行することによって、オペレーションのセマンティクスを保証した復元が可能となる面において寄与するところが大きい。

【０３００】なお、上記の処理機能は、コンピュータによって実現することができる。その場合、説明した処理内容は、コンピュータで読み取り可能な記録媒体に記録されたプログラムに記述されており、このプログラムをコンピュータで実行することにより、上記処理がコンピュータで実現される。コンピュータで読み取り可能な記録媒体としては、磁気記録装置や半導体メモリ等がある。市場へ流通させる場合には、ＣＤ－ＲＯＭ(Compact Disk Read Only Memory)やフロッピーディスク等の可搬型記録媒体にプログラムを格納して流通させたり、ネットワークを介して接続されたコンピュータの記憶装置に格納しておき、ネットワークを通じて他のコンピュータに転送することもできる。コンピュータで実行する際には、コンピュータ内のハードディスク装置等にプログラムを格納しておき、メインメモリにロードして実行する。

【０３０１】

【発明の効果】以上説明したように、第１の発明では、メタキャッシュ内のメタデータとともにメタデータがどのメタボリュームから取り出されたのかを示すメタデータ管理情報をログとして採取するようにしたため、保持されたログがどのメタボリュームのメタデータに関するログであるのかを管理することができる。その結果、複数のログボリュームにメタデータが格納されていても、ファイルシステムの不整合を修正することが可能となる。

【０３０２】また、第２の発明では、ログデータの有効範囲を管理するようにしたため、ファイルシステムを復元する際には、ログボリューム１２内の有効なログのみを用いて効率よく復元処理を行うことが可能となる。

【０３０３】また、第３の発明では、ログデータに対して付与するシーケンス番号の最大値を、システムの使用可能年数以上使い続けることができる値としたことで、常に昇順の採番が可能となり、ログボリュームのゼロクリアに伴う処理の遅延を避けることができる。

【０３０４】また、第４の発明では、メタデータが複数回更新される場合には、最終形態のみをログとして採取するようにしたため、ログデータが短縮されるとともに、ログデータの短縮に伴いファイルシステム復元時間の短縮が図れる。

【０３０５】また、第５の発明では、割り当て管理情報の一部の複製を獲得操作用管理情報とし、メタデータの獲得時には獲得操作用管理情報内から取得すべきメタデータを特定するが、解放時には獲得操作用管理情報の情報を更新しないようにしたため、メタデータの解放直後に別のトランザクションにより獲得されることがなくなる。その結果、システムダウン時に中途までしか終了していなかった解放トランザクションが解放したはずである領域は、解放される直前の状態のまま保全されることが保証される。

【０３０６】また、第６の発明では、獲得、解放操作のログとして、その操作対象となった情報のみを記録するようにしたため、ログ採取量が少量ですむ。また、第７の発明では、複数のログバッファを設け、さらにいくつか異なるサイズのログバッファを用意し、トランザクション毎のログをそのトランザクションに適したサイズのログバッファに格納するようにしたため、複数のトランザクションが並列実行される際のトランザクションの独立性を高めることができ、また、メモリ空間を有効に活用できる。

【０３０７】また、第８の発明では、１つのトランザクションのログがログバッファに収まらない場合に、中間ログとして完結されたログをログボリュームに書き出すようにしたため、部分的であるログを、ファイル復元時には１つのトランザクションのログと同値に扱うことが可能となり、ファイルシステムの復元時に望ましい状態に復元するのが容易となる。

【０３０８】また、第９の発明では、ログ採取に関するシステムの状態よってトランザクションの受け入れを制限するようにしたため、複数のトランザクションが並列実行されることによるメモリ枯渇等の障害の発生を防止することができる。

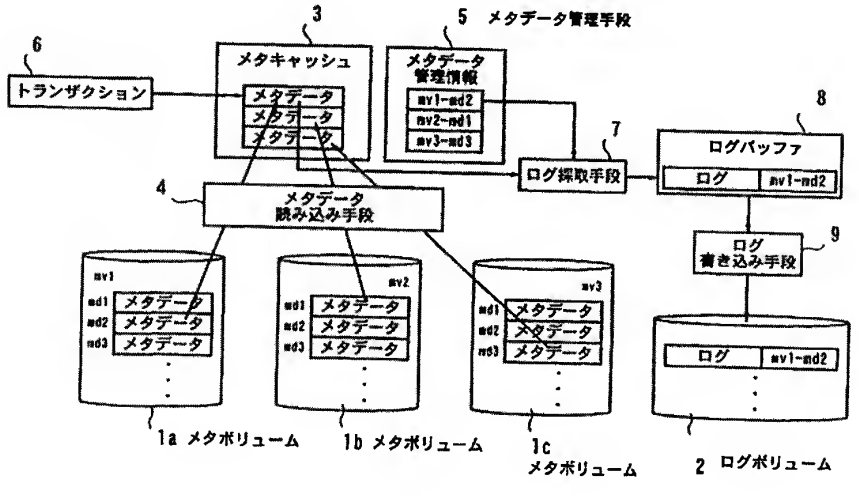
【図面の簡単な説明】

【図１】第１の発明の原理構成図である。
【図２】第２の発明の原理構成図である。
【図３】第３の発明の原理構成図である。
【図４】第４の発明の原理構成図である。
【図５】第５の発明の原理構成図である。
【図６】第６の発明の原理構成図である。
【図７】第７の発明の原理構成図である。
【図８】第８の発明の原理構成図である。
【図９】第９の発明の原理構成図である。
【図１０】本発明を適用するデータ処理装置のハードウェア構成図である。
【図１１】ファイルシステム上で動作するログ採取機能の構成図である。
【図１２】メタデータ管理情報を示す図である。
【図１３】ログバッファの形式を示す図である。
【図１４】有効範囲を説明する図である。
【図１５】ログ採取処理のフローチャートである。
【図１６】メタボリュームの割り当て管理状況を示す図である。
【図１７】ビットマップによるメタデータ獲得処理を示すフローチャートである。
【図１８】解放処理のフローチャートである。
【図１９】トランザクションの処理の獲得と終了の状況を示す図である。
【図２０】ログバッファへのログの格納状況を示す図である。
【図２１】ログ採取手順を示すフローチャートである。
【図２２】ファイルシステム復元処理を示すフローチャートである。
【図２３】新規トランザクションの受け入れ許否判定処理を示すフローチャートである。

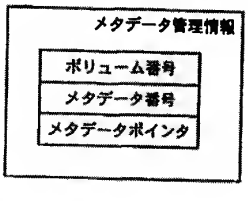
【符号の説明】

１ａ～１ｃ　メタボリューム
２　ログボリューム
３　メタキャッシュ
４　メタデータ読み込み手段
５　メタデータ管理情報
６　トランザクション
７　ログ採取手段
８　ログバッファ
９　ログ書き込み手段

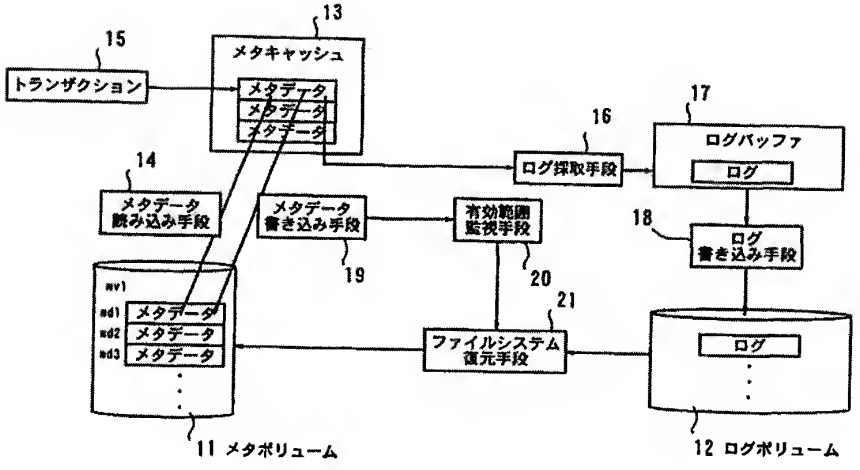
【図１】



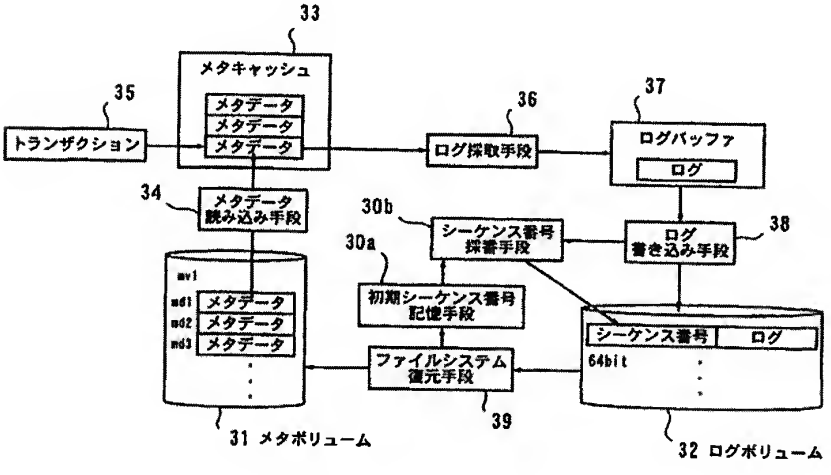
【図１２】



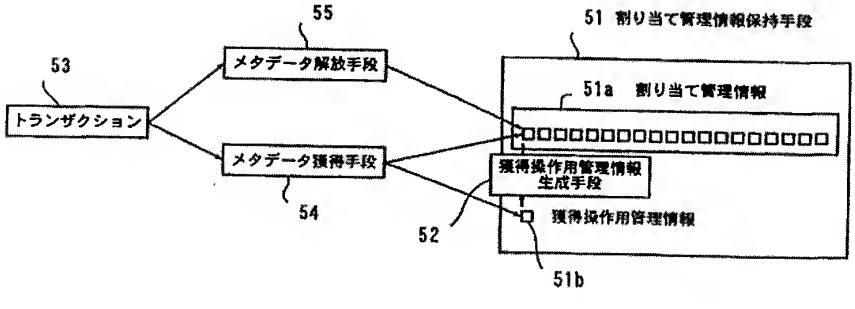
【図２】


13
メタキャッシュ
15
トランザクション
メタデータ
メタデータ
メタデータ
16
ログ採取手段
17
ログバッファ
ログ
14
メタデータ
読み込み手段
メタデータ
書き込み手段
有効範囲
監視手段
18
ログ
書き込み手段
19
20
21
mv1
md1
md2
md3
メタデータ
メタデータ
メタデータ
ファイルシステム
復元手段
ログ
11 メタボリューム
12 ログボリューム

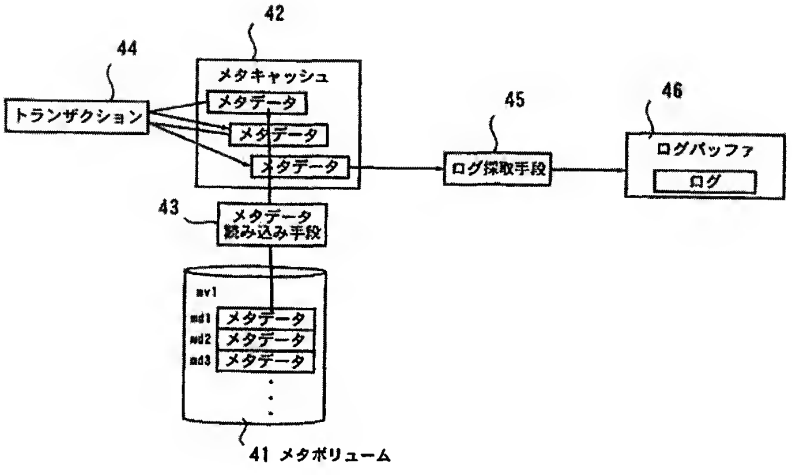
【図３】


33
メタキャッシュ
35
トランザクション
メタデータ
メタデータ
メタデータ
36
ログ採取手段
37
ログバッファ
ログ
34
メタデータ
読み込み手段
30b
シーケンス番号
採番手段
38
ログ
書き込み手段
30a
初期シーケンス番号
記憶手段
mv1
md1
md2
md3
メタデータ
メタデータ
メタデータ
シーケンス番号
ログ
64bit
ファイルシステム
復元手段
39
31 メタボリューム
32 ログボリューム

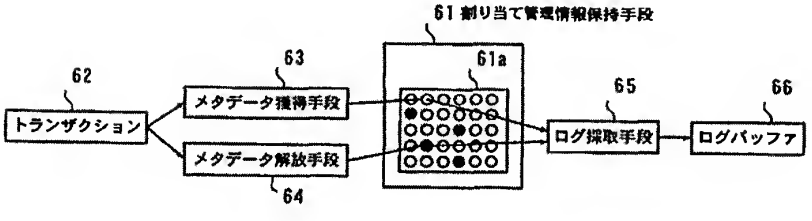
【図５】


55
51 割り当て管理情報保持手段
53
メタデータ解放手段
51a 割り当て管理情報
トランザクション
メタデータ獲得手段
獲得操作用管理情報
生成手段
54
52
獲得操作用管理情報
51b

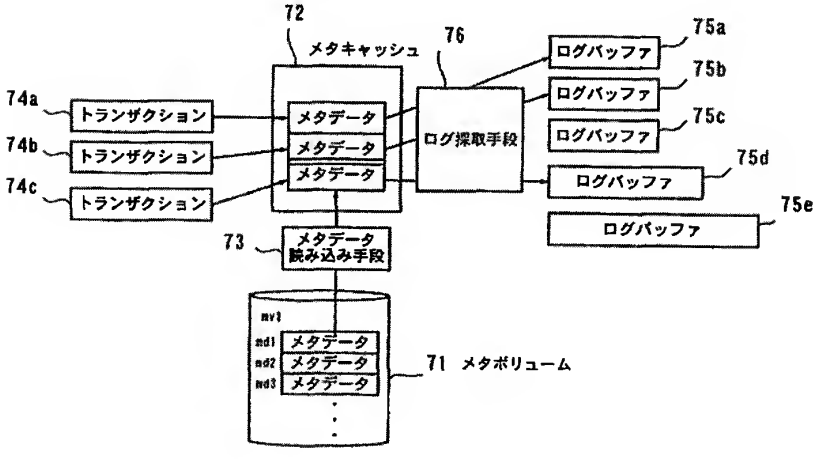
【図４】


42
44
メタキャッシュ
トランザクション
メタデータ
メタデータ
メタデータ
45
46
ログ採取手段
ログバッファ
ログ
43
メタデータ
読み込み手段
mv1
md1
メタデータ
md2
メタデータ
md3
メタデータ
41 メタボリューム

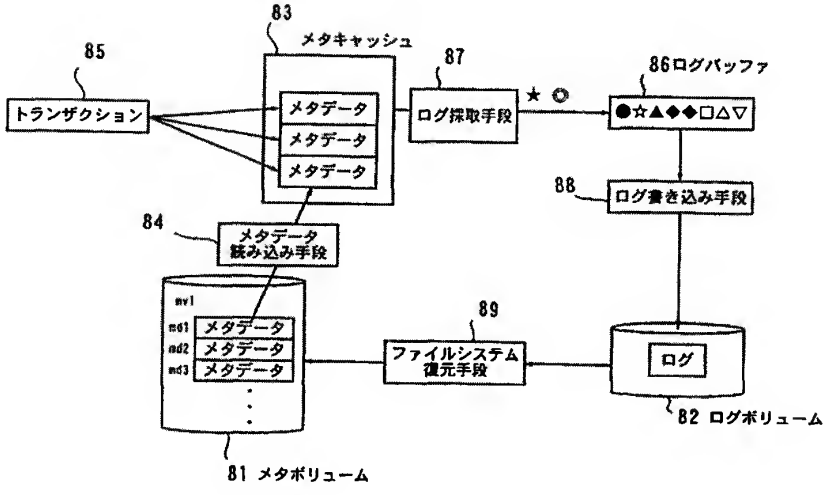
【図６】


61 割り当て管理情報保持手段
62
63
61a
65
66
トランザクション
メタデータ獲得手段
ログ採取手段
ログバッファ
メタデータ解放手段
64

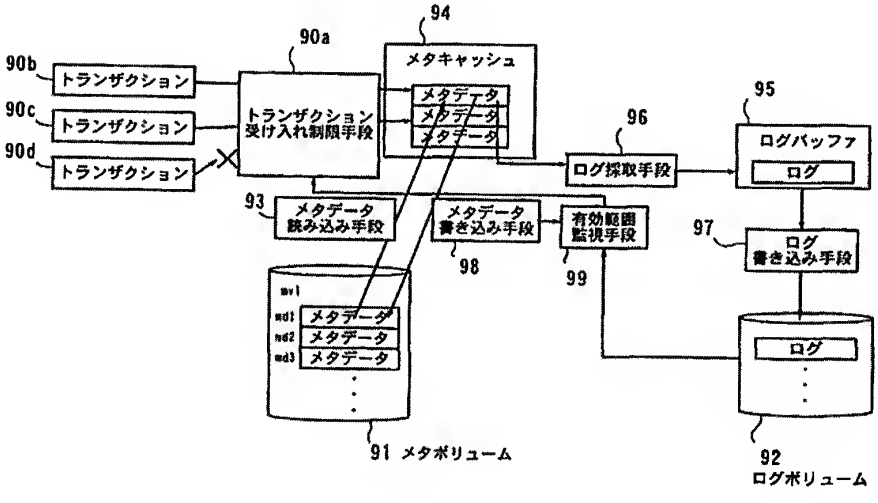
【図７】


72
メタキャッシュ
76
75a
ログバッファ
75b
ログバッファ
74a
トランザクション
メタデータ
74b
トランザクション
メタデータ
ログ採取手段
75c
ログバッファ
74c
トランザクション
メタデータ
75d
ログバッファ
75e
ログバッファ
73
メタデータ
読み込み手段
mv1
md1
メタデータ
md2
メタデータ
md3
メタデータ
71 メタボリューム

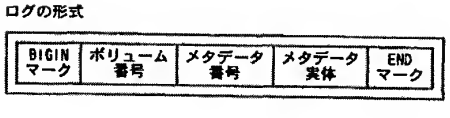
【図８】


83
メタキャッシュ
85
トランザクション
メタデータ
メタデータ
メタデータ
87
ログ採取手段
86ログバッファ
●☆▲◆◆□△▽
88
ログ書き込み手段
84
メタデータ
読み込み手段
mv1
md1 メタデータ
md2 メタデータ
md3 メタデータ
89
ファイルシステム
復元手段
ログ
82 ログボリューム
81 メタボリューム

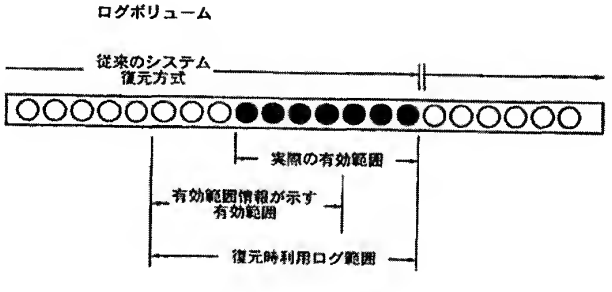
【図９】


94
90a
メタキャッシュ
90b
トランザクション
90c
トランザクション
90d
トランザクション
トランザクション
受け入れ制限手段
メタデータ
メタデータ
メタデータ
95
96
ログバッファ
ログ
ログ採取手段
93
メタデータ
読み込み手段
メタデータ
書き込み手段
98
有効範囲
監視手段
99
97
ログ
書き込み手段
mv1
md1 メタデータ
md2 メタデータ
md3 メタデータ
ログ
91 メタボリューム
92
ログボリューム

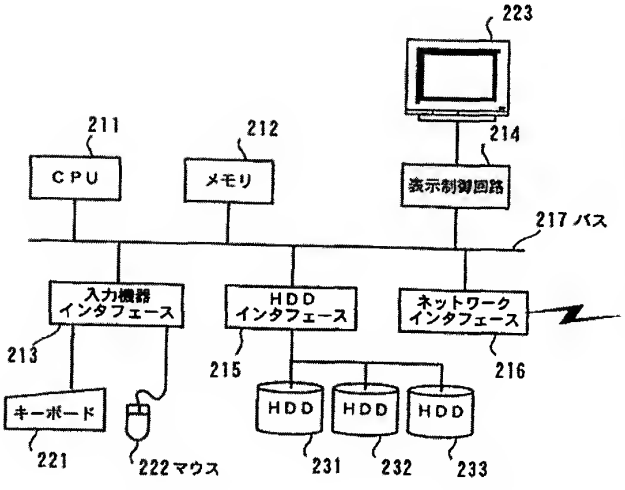
【図１３】


ログの形式
BIGIN マーク
ボリューム 番号
メタデータ 番号
メタデータ 実体
END マーク

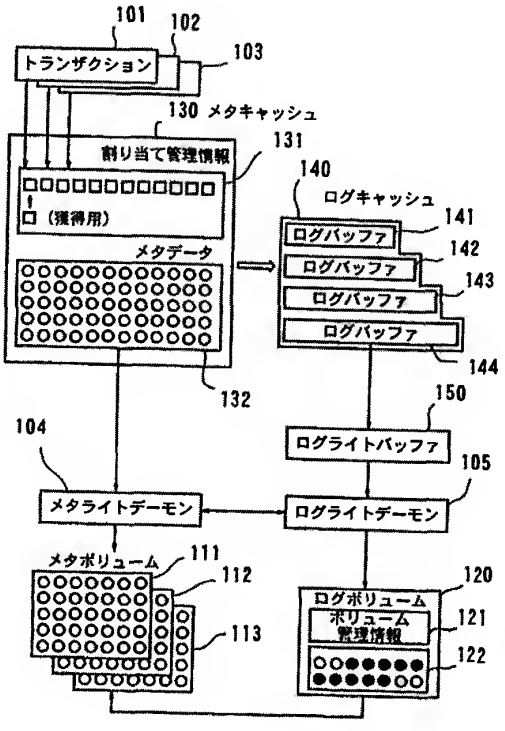
【図１４】


ログボリューム
従来のシステム 復元方式
実際の有効範囲
有効範囲情報が示す 有効範囲
復元時利用ログ範囲

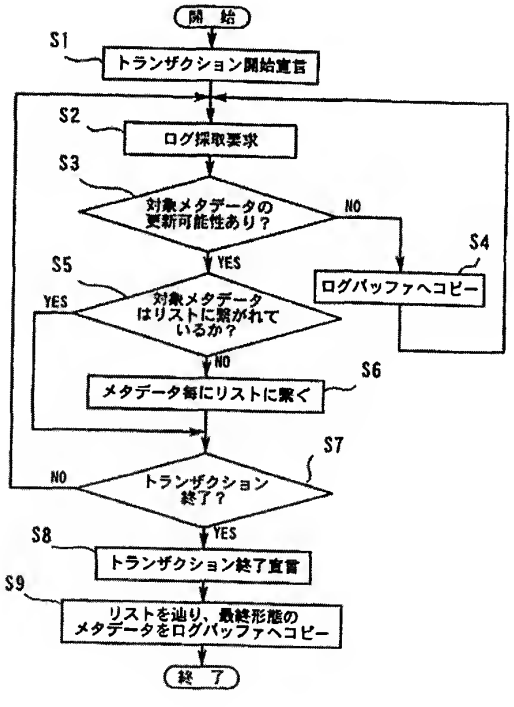
【図１０】



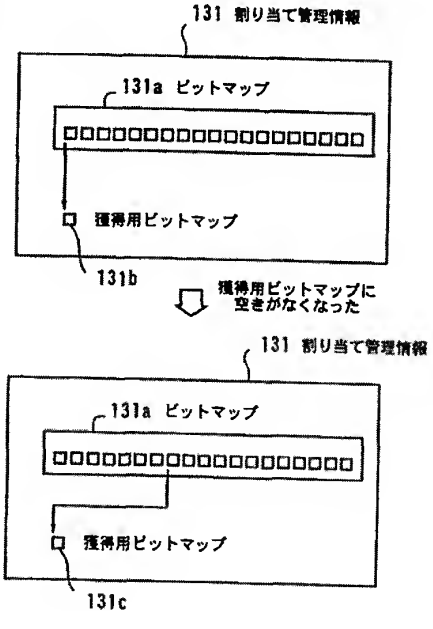
【図１１】



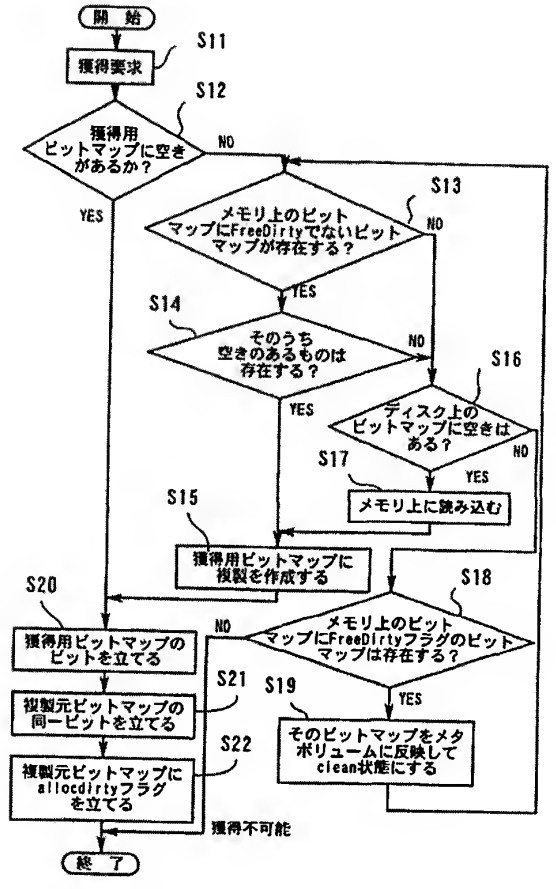
【図１５】



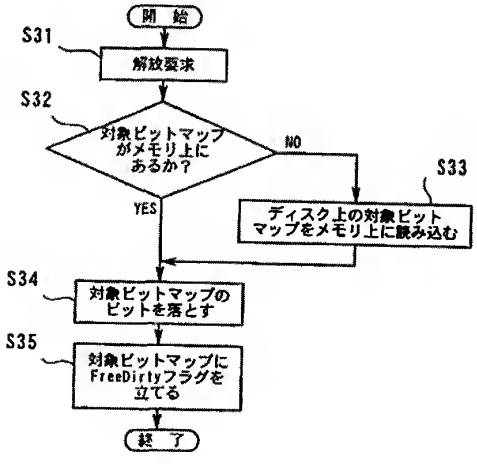
【図１６】



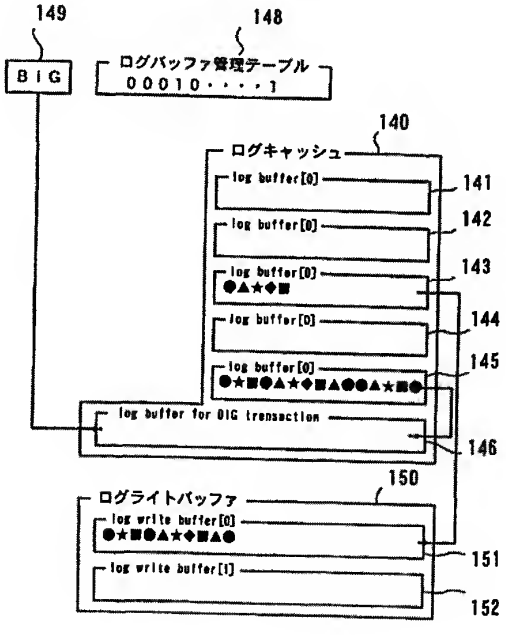
【図１７】



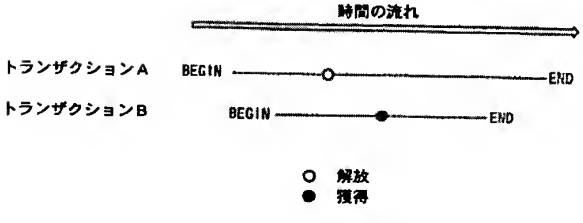
【図１８】



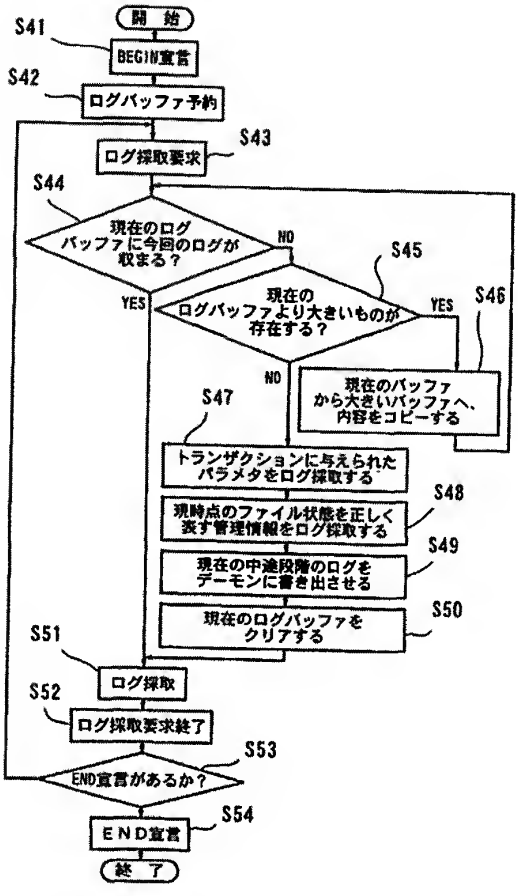
【図２０】



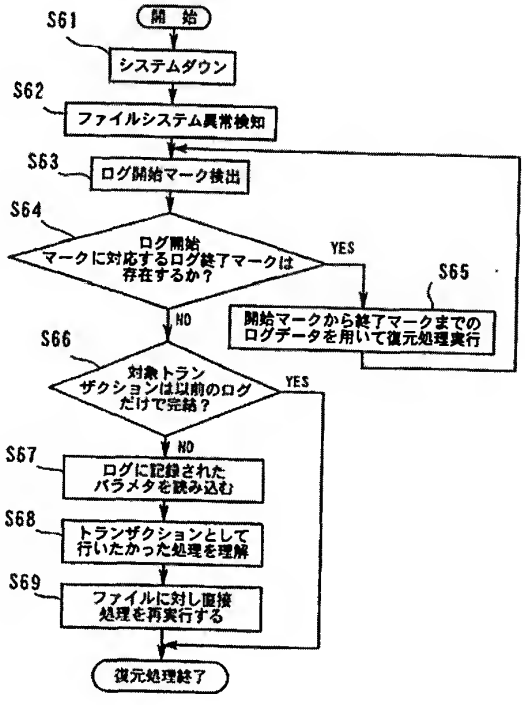
【図１９】



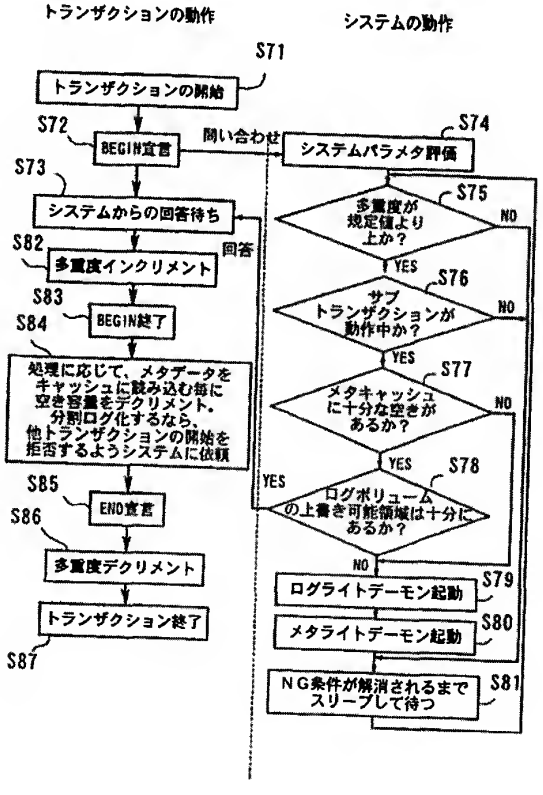
【図２１】


開　始
S41
BEGIN宣言
S42
ログバッファ予約
S43
ログ採取要求
S44
現在のログバッファに今回のログが収まる？
NO
S45
現在のログバッファより大きいものが存在する？
YES
YES
S46
現在のバッファから大きいバッファへ、内容をコピーする
NO
S47
トランザクションに与えられたパラメタをログ採取する
S48
現時点のファイル状態を正しく表す管理情報をログ採取する
S49
現在の中途段階のログをデーモンに書き出させる
S50
現在のログバッファをクリアする
S51
ログ採取
S52
ログ採取要求終了
S53
END宣言があるか？
S54
END宣言
終　了

【図２２】


開　始
S61
システムダウン
S62
ファイルシステム異常検知
S63
ログ開始マーク検出
S64
ログ開始マークに対応するログ終了マークは存在するか？
YES
S65
開始マークから終了マークまでのログデータを用いて復元処理実行
NO
S66
対象トランザクションは以前のログだけで完結？
YES
NO
S67
ログに記録されたパラメタを読み込む
S68
トランザクションとして行いたかった処理を理解
S69
ファイルに対し直接処理を再実行する
復元処理終了

【図２３】


トランザクションの動作
システムの動作
S71
トランザクションの開始
S72
BEGIN宣言
問い合わせ
S74
システムパラメタ評価
S73
システムからの回答待ち
回答
S75
多重度が規定値より上か？
NO
YES
S82
多重度インクリメント
S76
サブトランザクションが動作中か？
NO
YES
S83
BEGIN終了
S84
処理に応じて、メタデータをキャッシュに読み込む毎に空き容量をデクリメント。分割ログ化するなら、他トランザクションの開始を拒否するようシステムに依頼
S77
メタキャッシュに十分な空きがあるか？
NO
YES
S78
ログボリュームの上書き可能領域は十分にあるか？
YES
NO
S85
END宣言
S86
多重度デクリメント
S79
ログライトデーモン起動
S80
メタライトデーモン起動
トランザクション終了
S87
NG条件が解消されるまでスリープして待つ
S81

【手続補正書】
【提出日】平成１２年２月２２日（２０００．２．２２）
【手続補正１】
【補正対象書類名】明細書
【補正対象項目名】全文
【補正方法】変更
【補正内容】
【書類名】　　明細書
【発明の名称】　データ処理装置及び記録媒体
【特許請求の範囲】
【請求項１】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリュームと、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段と、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段と、
前記ログ採取手段が採取した情報を保持するログバッファと、
前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、
を有することを特徴とするデータ処理装置。
【請求項２】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、
前記ログ採取手段が採取した情報を保持するログバッファと、
前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、
前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、
前記メクデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段と、
ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、
を有することを特徴とするデータ処理装置。
【請求項３】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、
前記ログ採取手段が採取した情報を保持するログバッファと、
前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、
前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、
前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段と、
前記ログ書き込み手段がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶

手段に格納されたシーケンス番号を基準として採番するシーケンス番号採番手段と、
を有することを特徴とするデータ処理装置。
【請求項４】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段と、
を有することを特徴とするデータ処理装置。
【請求項５】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段と、
前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作用管理情報とするとともに、前記獲得操作用管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作用管理情報とする獲得操作用管理情報生成手段と、
メタデータの獲得及び解放要求を出力するトランザクションと、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作用管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作用管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段と、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、
を有することを特徴とするデータ処理装置。
【請求項６】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段と、
メタデータの獲得及び解放要求を出力するトランザクションと、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記割り当て管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段と、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、
前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段と、
を有することを特徴とするデータ処理装置。
【請求項７】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクションと、
ログをトランザクション毎に保持する、サイズの異なる複数のログバッファと、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段と、
を有することを特徴とするデータ処理装置。
【請求項８】　前記ログ採取手段は、最初にログを格納する場合には、トランザクションの内容によって予想される処理に適した大きさの前記ログバッファに格納し、格納対象となる前記ログバッファの記憶容量が不足してきたら、より大きな記憶容量のログバッファへログを移し替え、以後、より大きな記憶容量のログバッファをログの格納対象とすることを特徴とする請求項７記載のデータ処理装置。
【請求項９】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
ログをトランザクション毎に保持するログバッファと、

前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段と、
前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段と、
を有することを特徴とするデータ処理装置。
【請求項１０】　前記ログ書き込み手段は、前記中間ログに対して、トランザクションを実行するのに必要とされたパラメタに関する情報を付加し、
ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログを用いて、前記メタボリューム内のメタデータの不整合を修正するとともに、前記中間ログを発見すると、前記中間ログに含まれたパラメタを用いて、前記トランザクションを再実行させるファイルシステム復元手段をさらに有することを特徴とする請求項９記載のデータ処理装置。
【請求項１１】　ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクションと、
前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ許否を判断するトランザクション受け入れ判断手段と、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、
前記ログ採取手段が採取した情報を保持するログバッファと、
前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、
を有することを特徴とするデータ処理装置。
【請求項１２】　前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、
前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段とをさらに有し、
前記トランザクション受け入れ判断手段は、前記有効範囲監視手段によって有効なログとされたログがログボリューム中に占める割合が一定値以上である間は、前記トランザクションの受け入れを拒絶することを特徴とする請求項１１記載のデータ処理装置。
【請求項１３】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段、
前記ログ採取手段が採取した情報を保持するログバッファ、
前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。
【請求項１４】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記メタキャッシュ内で変更されたメタデータの内容を

ログとして採取するログ採取手段、
前記ログ採取手段が採取した情報を保持するログバッファ、
前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段、
前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段、
前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段、
ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１５】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段、
前記ログ採取手段が採取した情報を保持するログバッファ、
前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段、
前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段、
前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段、
前記ログ書き込み手段がログの書き込みを行う際に、システムの使用可能年数以上使い続けることができる値を最大値としたシーケンス番号を昇順で採番し、書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶手段に格納されたシーケンス番号から昇順で採番するシーケンス番号採番手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１６】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１７】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段、
前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作用管理情報とするとともに、前記獲得操作用管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作用管理情報とする獲得操作用管理情報生成手段、
メタデータの獲得及び解放要求を出力するトランザクション、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作用管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作用管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１８】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、

メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段、

メタデータの獲得及び解放要求を出力するトランザクション、

前記トランザクションによりメタデータの獲得要求が出された場合には、前記割り当て管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段、

前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段、

前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項１９】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、

ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、

メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、

メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、

前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクション、

ログをトランザクション毎に保持する、サイズの異なる複数のログバッファ、

前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項２０】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、

ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、

メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、

メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、

メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、

前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、

ログをトランザクション毎に保持するログバッファ、

前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段、

前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項２１】　ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、

ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、

メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、

メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、

メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、

前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクション、

前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ許否を判断するトランザクション受け入れ判断手段、

前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段、

前記ログ採取手段が採取した情報を保持するログバッファ、

前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【０００１】
【発明の属する技術分野】本発明はファイルシステムの修正機能を有するデータ処理装置及び記録媒体に関し、特にログを用いてファイルシステムの不整合の修正を行うデータ処理装置及び記録媒体に関する。
【０００２】
【従来の技術】コンピュータシステムを運用していると、何らかの理由によりシステムがダウンする場合がある。突然のシステムダウンが発生すると、ファイルシステムの不整合が生じる。そこで、システムダウン後のブート時には、従来であればファイルシステム全体を走査し、その矛盾点の検出を行う。矛盾点が発見されたら、場合に応じた変更をファイルシステムに加えることによって、ファイルシステムの整合性を回復していた。ところが、ファイルシステム全体を走査するには多くの時間が必要であり、結果としてシステムダウン後の復旧の遅れを招いていた。
【０００３】そこで近代のＵＮＩＸオペレーティングシステム（ＯＳ）のような多くのコンピュータＯＳのファイルシステムでは、ファイルシステムオペレーションにおいてファイルシステムに保存されているデータが更新される度に、その更新情報をログ（ジャーナル）として採取している。ファイルシステムオペレーション時に更新情報のログ採取を行っておくことにより、システムダウン後のブート時のファイルシステム整合性チェックのフェーズでは、残されたログを順次走査し、対応する領域へアップデートすることによって、ファイルシステムの整合性が保証される。その結果、システムのダウン時間が短縮される。
【０００４】
【発明が解決しようとする課題】しかし、ログ機構を導入することにより、次に挙げる問題が生じていた。
（１）第１の問題点
ファイルを管理するメタデータ（二次記憶装置に格納されたファイルを管理するためのデータ）は、ファイルシステムオペレーション時に二次記憶装置からメモリへ読み込まれ、メモリ内において操作される。その後、所定のタイミングで二次記憶装置へとその更新内容が反映される。ログ機構の導入時には、その二次記憶装置への反映前に更新内容をログ専用の二次記憶装置へと記録する必要がある。
【０００５】しかし、大規模ファイルシステムに対応するために、複数の二次記憶装置がメタデータに割り当てられ、異なる二次記憶装置に割り当てられたメタデータを１つのファイルシステムオペレーションが操作する場合がある。このファイルシステムオペレーションのログを採取する際に、メモリ内のメタデータだけをログとして記録していたのでは、残されたログからメタデータ毎に異なる二次記憶装置を検索するのに時間を消費し、ログ機構導入による最大のメリットである、ファイルシステム復元の時間短縮に悪い影響を与える。
【０００６】（２）第２の問題点
同様にファイルシステム復元の時間短縮に悪い影響を与える要因として、有限なサイズしかないログ専用の二次記憶装置全体をファイルシステム復元時に探索することが考えられる。
【０００７】ログ機構の導入はファイルシステムオペレーションを細分化したトランザクションの完了を常に保証しつつ動作するため、ログ機構を導入した多くのファイルシステムはトランザクションにシーケンス番号を与え、ファイルシステム復元時にはシーケンス番号をもとに最古のトランザクションを同定する。そして、最古のトランザクションからログリプレイと呼ばれるログを用いたファイルシステム復元操作を開始する。
【０００８】ここで、有限サイズのログ専用の二次記憶装置内には、ファイルシステムの整合性を復元するために欠かせないログが確かに存在する可能性があるが、有限なサイズを有効利用するためにログ専用の二次記憶装置は過去のログを上書きしてサイクリックに利用しなければならない。このような処理を行うには、ある程度以上古いログが必ず不必要となっていることが前提となる。従って、多くのログの内容は既に不必要となっている。すなわち、ログ専用の二次記憶装置全体を探索あるいは反映するのは非効率的である。
【０００９】（３）第３の問題点
最古のトランザクションを特定するためのシーケンス番号は単調増加であることが要求され、運用途中にオーバーフローによってゼロに戻されてしまうことは許されない。これを回避するために、ファイルシステム復元作業終了時、またはオーバーフロー直前にログ専用の二次記憶装置全体をゼロクリアし、再度シーケンス番号ゼロから順にトランザクションを処理するのが一般的である。しかし、ログ専用の二次記憶装置をゼロクリアするためには多くの時間が必要となる。少なくともシステムの運用を一時停止する必要があり、サーバ装置などによるサービスの提供を停止せざるを得なくなってしまう。
【００１０】以上の（１）～（３）の課題はファイルシステム復元時の問題であるが、ログ機構を導入することは通常の運用時にも問題を引き起こす可能性を持っている。ログ機構は、メモリ上にログをスプールする手法や、ログ専用の二次記憶装置として用いられるディスクの特性を考慮したシーケンシャルアクセスなど、高速化のための条件は整えられているが、ログの採取の仕方を熟考しなければ、ログ採取に伴う性能劣化は非常に大きいものとなりうる。
【００１１】（４）第４の問題点
単一のトランザクション内で、同一データを複数回更新することは度々あるが、その都度、その同一データに対するログを採取したのではメモリが不足し、二次記憶装置へのＩ／Ｏ量が増加する。

【００１２】（５）第５の問題点
ログ機構の導入はトランザクションの順序性を保証し、終了したトランザクションのログを順に採取することを要求するために、あるトランザクションが操作したログ対象データを他のトランザクションが操作することが一般的に不可能な状況となりうる。ここで、個々のファイルの内容を対象とするトランザクションについては、ファイル単位に排他制御を行うことによって、複数のトランザクションが並列実行することは比較的容易である。しかしながら、個々のファイルによらないもの、特に領域の割り当て情報を操作する場合には、並列実行がきわめて難しくなる。
【００１３】領域の獲得・解放処理が並列に動作する場合を考えると、それぞれが獲得、解放のログを採取する。同一の管理情報（ここでは、ビットマップを考える）によって管理される領域の獲得・解放処理であった場合には、同一の管理情報が、解放された状態、獲得された状態でログに記録されることになる。
【００１４】ファイルシステムの復元処理を行わなければならないようなシステムダウンが発生するタイミングによっては、このように別トランザクションの更新情報を含むログの採取方式では様々な問題が生じる。
【００１５】Ａというトランザクションが解放処理、Ｂというトランザクションが獲得処理を行う場合を考える。ここで、トランザクションＡの解放処理はトランザクションＢの獲得処理よりも先に行われ、かつ、トランザクションＡはトランザクションＢよりも後に終わる場合を例に挙げる。
【００１６】このとき、トランザクションＡが解放した領域をトランザクションＢが獲得してしまう場合が考えられる。トランザクションＢが先に終了することから、残されたログには、まず獲得処理が記録され、次に解放処理が記録される。
【００１７】上記の場合のトランザクションＢのログだけが記録されており、それを復元に用いた場合には、本来行われたはずである解放処理の記録が残されていないことから、該当領域が二重に獲得された状態となってしまう。トランザクションＡのログまで記録されており、復元に用いられると、トランザクションＢが利用している領域がトランザクションＡの解放処理のログによって解放されている状態となってしまう。いずれも本来の状態とは異なっており、避けなければならない。しかしながら、これを回避するために並列実行を制限することは、マルチタスクを実現したＯＳ上のファイルシステムの速度性能の低下に与える影響が非常に大きい。
【００１８】（６）第６の問題点
既に述べたようにログ機構の第一の目的としてファイルシステムを復元するために費やす時間の短縮が挙げられるが、そのためにトランザクションの独立性を疎かにしたり、中途半端な状態での整合性回復によりあたかも正常に動作しているかのように振る舞うファイルシステムが多く見受けられる。
【００１９】従来のメタデータ管理方式のように、ログを記録するためのメモリ空間をファイルシステム全体で１つのキャッシュメモリを共有していたのではトランザクション毎の独立性を保つのが難しく、他のトランザクションによる更新情報が１つのトランザクションのログとして記録されてしまう可能性が高い。特にファイルに対する排他で制御しきれない割り当て管理情報については前述の通りである。
【００２０】（７）第７の問題点
トランザクション毎に必要とするログのサイズが異なることから、複数のログバッファとしてログを記録するためのメモリをトランザクション毎に割り当て、管理する場合に、全てのメモリサイズが同一である必要はない。例えば、ファイルの参照時刻を更新するだけのトランザクションが残すログのサイズは大変小さく、巨大なデータ書き込み要求のトランザクションは必然的にそれだけ大きいログサイズとなる。
【００２１】（８）第８の問題点
トランザクション毎に残すログサイズの違いを考慮して、限られたメモリ空間の有効利用を試みても、キャッシュメモリサイズは残されるログサイズに比較して、やはり小さい。
【００２２】（９）第９の問題点
「第８の問題点」の解決策として、１つのトランザクションを分割し、中途の状態のログを出力することが考えられる。しかし、一般にファイルを管理するメタデータは、トランザクションが中途の状態ではやはり中途の状態であり、そのままログに記録したところで、そのログを用いて復元されるファイルシステムは中途の状態にしかなり得ない。これではオペレーションのセマンティクスを保証した復元とはなり得ない。
【００２３】（１０）第１０の問題点
「第９の問題点」で示した中途の状態でのトランザクションの中断はファイルシステムにとって危険な状態といえる。ログ機構の導入は、採取したログをログボリュームに反映するまでは該当メタデータもキャッシュメモリから追い出せないことを意味する。ログがキャッシュメモリに入りきらないほど巨大となっているときには、メタデータを管理するキャッシュメモリの利用率も高くなっていることが考えられる。ログを出力するまでメタデータをキャッシュメモリから追い出せないため、このような状態で多くのトランザクションが並列実行されると、メモリ枯渇によるハングアップ状態に陥ることも考えられる。
【００２４】（１１）第１１の問題点
第１０の問題点と同様の資源枯渇はログ記録用の二次記憶装置についても言える。キャッシュメモリに比較すれば巨大な二次記憶装置についても、メタデータ記録用の

二次記憶装置へのＩ／Ｏを削減するために、多くのログを有効なログとして残しておけば、それは利用可能な領域の減少を引き起こす。その際に多数のトランザクションの並列実行を許すことはメタデータキャッシュ枯渇、ログキャッシュ枯渇、ログ記録用二次記憶装置枯渇、などを誘発する可能性がある。

【００２５】本発明はこのような点に鑑みてなされたものであり、ファイルシステム修復処理の効率化を図ったデータ処理装置を提供することを目的とする。

【００２６】

【課題を解決するための手段】本発明では上記課題を解決するための第１の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段と、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【００２７】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。その際、読み込んだメタデータがどのメタボリュームから読み込まれたものであるのかが、メタデータ管理手段によって管理される。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。この際、メタデータが格納されていたメタボリュームの識別情報をも採取する。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。

【００２８】また、上記課題を解決する第２の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログボリュームの領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段と、ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、を有することを特徴とするデータ処理装置が提供される。

【００２９】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。その後、メタデータ書き込み手段により、メタキャッシュ内のメタデータがメタボリューム内に格納される。その書き込み動作は、有効範囲監視手段で監視されており、変更内容がメタボリュームに反映されていないメタデータに対応するログボリューム内のログが、有効なログとして指定される。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段により、ログボリュームに格納されたログの中で、有効範囲監視手段により有効なログとして指定されているログのみを用いて、メタボリューム内のメタデータの不整合が修正される。

【００３０】また、上記課題を解決する第３の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となる

メタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段と、前記ログ書き込み手段がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶手段に格納されたシーケンス番号を基準として採番するシーケンス番号採番手段と、を有することを特徴とするデータ処理装置が提供される。

【００３１】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。その際、シーケンス番号採番手段により、シーケンス番号が昇順で採番され、書き込むべきログに付与される。ファイルシステムに不整合が発生すると、ファイルシステム復元手段により、ログボリュームに残されたログを用いてメタデータの不整合が修正される。このとき、修正に用いられた最後のログのシーケンス番号が初期シーケンス番号記憶手段に記憶される。その後、ログ書き込み手段によりログバッファ内の情報がログボリュームに書き込まれると、シーケンス番号採番手段により、初期シーケンス番号記憶手段に格納されたシーケンス番号を基準としてシーケンス番号が採番され、書き込むべきログに付与される。

【００３２】また、上記課題を解決する第４の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【００３３】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新する。すると、ログ採取手段により、トランザクションの種別が判断され、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、メタキャッシュ内で変更されたメタデータの最終形態のみがログとして採取される。

【００３４】また、上記課題を解決する第５の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段と、前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作用管理情報とするとともに、前記獲得操作用管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作用管理情報とする獲得操作用管理情報生成手段と、メタデータの獲得及び解放要求を出力するトランザクションと、前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作用管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作用管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段と、前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、を有することを特徴とするデータ処理装置が提供される。

【００３５】このようなデータ処理装置によれば、獲得操作用管理情報生成手段により、割り当て管理情報の一部領域の複製が生成され、獲得操作用管理情報とされる。トランザクションにより獲得要求があると、メタデータ獲得手段により、獲得操作用管理情報内の未獲得のメタデータが獲得される。すると、獲得操作用管理情報と割り当て管理情報との内容が更新される。また、トランザクションよりメタデータの解放要求があると、メタデータ解放手段により該当するメタデータの解放処理が行われる。この際、割り当て管理情報の内容のみが更新される。ここで、獲得操作用管理情報内に未獲得のメタデータがなくなると、割り当て管理情報内の別の領域の

複製が獲得操作用管理情報とされる。

【００３６】また、上記課題を解決する第６の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段と、メタデータの獲得及び解放要求を出力するトランザクションと、前記トランザクションによりメタデータの獲得要求が出された場合には、前記割り当て管理情報保持手段の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段と、前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【００３７】このようなデータ処理装置によれば、トランザクションからメタデータの獲得要求が出されると、メタデータ獲得手段によって未獲得のメタデータの１つが割り当て管理情報内から獲得され、そのメタデータが獲得済みの状態とされる。また、トランザクションからメタデータの解放要求が出されると、メタデータ解放手段によって該当するメタデータが未獲得の状態に変更される。そして、ログ採取手段により、割り当て管理情報内のメタデータ獲得手段及びメタデータ解放手段によって変更された部分の情報がログとして採取される。

【００３８】また、上記課題を解決する第７の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクションと、　ログをトランザクション毎に保持する、サイズの異なる複数のログバッファと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【００３９】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新する。すると、ログ採取手段により、メタキャッシュ内で変更されたメタデータがログとして採取され、トランザクション毎のログバッファに保持される。

【００４０】また、上記課題を解決する第８の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、ログをトランザクション毎に保持するログバッファと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段と、前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【００４１】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログバッファに格納しきれなくなるか、トランザクションが終了すると、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。ログバッファに格納しきれなくなった場合には、ログバッファの内容を完結したログに加工され、中間ログとしてログボリュームに格納される。

【００４２】また、上記課題を解決する第９の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクションと、前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ許否を判

断するトランザクション受け入れ判断手段と、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【００４３】このようなデータ処理装置によれば、トランザクションから開始要求が出されると、トランザクション受け入れ制限手段が受け入れの許否を判断する。その後、メタデータ書き込み手段によるメタデータの書き込みが進み、有効なログの割合が減少したら、その時点でトランザクションの開始要求を許可する。

【００４４】

【発明の実施の形態】以下、本発明の実施の形態を図面を参照して説明する。図１は、第１の発明の原理構成図である。この第１の発明は、複数の二次記憶装置がメタデータに割り当てられている場合におけるファイルシステム不整合修復時間の短縮を図るものである。

【００４５】このデータ処理装置には、二次記憶装置としてメタボリューム１ａ，１ｂ，１ｃとログボリューム２とが設けられている。各メタボリューム１ａ，１ｂ，１ｃには、ファイルを管理するためのメタデータが記憶されている。また、ログボリューム２には、メタデータの更新結果であるログが記憶されている。また、主記憶装置内にはメタキャッシュ３が設けられている。メタキャッシュ３は、メタデータを記憶するための主記憶装置内の記憶領域である。メタデータ読み込み手段４は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ３へと読み込む。メタデータ管理手段５は、メタキャッシュ３に読み込まれたメタデータが格納されていたメタボリューム１ａ，１ｂ，１ｃの識別情報を管理する。トランザクション６は、メタキャッシュ３内に読み込まれたメタデータの内容を更新する。ログ採取手段７は、メタキャッシュ３内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリューム１ａ，１ｂ，１ｃの識別情報を採取する。ログバッファ８は、ログ採取手段７が採取した情報を保持する。ログ書き込み手段９は、ログバッファ８が保持する情報を、適宜ログボリューム２に格納する。

【００４６】このようなデータ処理装置によれば、トランザクション６がメタデータの更新をする際には、まず、メタデータ読み込み手段４によりメタボリューム１ａ～１ｃ内の必要なメタデータがメタキャッシュ３に読み込まれる。その際、読み込んだメタデータがどのメタボリュームから読み込まれたものであるのかが、メタデータ管理手段５によって管理される。ここで、トランザクション６がメタデータの内容を更新すると、ログ採取手段７によって更新後のメタデータがログとして採取される。この際、メタデータが格納されていたメタボリューム１ａ～１ｃの識別情報をも採取する。採取した情報は、ログバッファ８に保持される。そして、ログ書き込み手段９により、ログバッファ８内の情報がログボリューム２に書き込まれる。

【００４７】これにより、ログボリューム２に保持されたログがどのメタボリューム１ａ～１ｃのメタデータに関するログであるのかを管理することができる。その結果、複数のメタボリューム１ａ～１ｃにメタデータが格納されていても、ファイルシステムの不整合を修正することが可能となる。

【００４８】図２は、第２の発明の原理構成図である。第２の発明は、ログの有効範囲を監視することで、ファイルシステム復元時の効率化を図ったものである。第２の発明は、以下のような要素で構成される。

【００４９】メタボリューム１１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム１２は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ１３は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段１４は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ１３へと読み込む。トランザクション１５は、メタキャッシュ１３内に読み込まれたメタデータの内容を更新する。ログ採取手段１６は、メタキャッシュ１３内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取する。ログバッファ１７は、ログ採取手段１６が採取した情報を保持する。ログ書き込み手段１８は、ログボリューム１２内の領域を定期的に循環するようにして、ログバッファ１７が保持する情報をログボリューム１２に格納する。メタデータ書き込み手段１９は、メタキャッシュ１３内のメタデータをメタボリューム１１内に格納する。有効範囲監視手段２０は、メタデータ書き込み手段１９による書き込み動作を監視しており、変更内容がメタボリューム１１に反映されていないメタデータに対応するログボリューム１２内のログを、有効なログとして指定する。ファイルシステム復元手段２１は、ファイルシステム復元要求を受け取ると、ログボリューム１２に格納されたログの中で、有効範囲監視手段２０により有効なログとして指定されているログのみを用いて、メタボリューム１１内のメタデータの不整合を修正する。なお、有効範囲監視手段２０による有効なログの指定方法としては、例えば、有効範囲を示す情報を不揮発性の記録媒体（ログボリューム１２等）に記録することができる。この場合、ファイルシステム復元手段２１は、有効範囲が記録された不揮発性の記録媒体内の情報を読みとることで、有効なログと指定されているログを認識できる。

【００５０】このようなデータ処理装置によれば、トラ

ンザクション１５がメタデータの更新をする際には、まず、メタデータ読み込み手段１４によりメタボリューム１１内の必要なメタデータがメタキャッシュ１３に読み込まれる。ここで、トランザクション１５がメタデータの内容を更新すると、ログ採取手段１６によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ１７に保持される。そして、ログ書き込み手段１８により、ログバッファ１７内の情報がログボリューム１２に書き込まれる。その後、メタデータ書き込み手段１９により、メタキャッシュ１３内のメタデータがメタボリューム１１内に格納される。その書き込み動作は、有効範囲監視手段２０で監視されており、変更内容がメタボリューム１１に反映されていないメタデータに対応するログボリューム１２内のログが、有効なログとして指定される。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段２１により、ログボリューム１２に格納されたログの中で、有効範囲監視手段２０により有効なログとして指定されているログを用いて、メタボリューム１１内のメタデータの不整合が修正される。

【００５１】これにより、ファイルシステムを復元する際には、ログボリューム１２内の有効なログのみを用いて効率よく復元処理を行うことが可能となる。図３は、第３の発明の原理構成図である。第３の発明は、ログに付加するシーケンス番号のデータサイズを十分大きなものとし、シーケンス番号を常に昇順で使い続けることができる（ゼロクリアが不要となる）ようにしたものである。第３の実施の形態は、以下のような要素で構成される。

【００５２】メタボリューム３１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム３２は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ３３は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段３４は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ３３へと読み込む。トランザクション３５は、メタキャッシュ３３内に読み込まれたメタデータの内容を更新する。ログ採取手段３６は、メタキャッシュ３３内で変更されたメタデータの内容をログとして採取する。ログバッファ３７は、ログ採取手段３６が採取した情報を保持する。ログ書き込み手段３８は、ログバッファが保持する情報を前記ログボリュームに格納する。ファイルシステム復元手段３９は、ログボリューム３２に格納されたログを用いてメタボリューム３１内のメタデータの不整合を修正する。初期シーケンス番号記憶手段３０ａは、ファイルシステム復元手段３９がメタボリューム３１内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する。シーケンス番号採番手段３０ｂは、ログ書き込み手段３８がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、ファイルシステム復元手段３９がメタボリューム３１内のメタデータの不整合を修正した直後には、初期シーケンス番号記憶手段３０ａに格納されたシーケンス番号を基準として採番する。

【００５３】このようなデータ処理装置によれば、トランザクション３５がメタデータの更新をする際には、まず、メタデータ読み込み手段３４によりメタボリューム３１内の必要なメタデータがメタキャッシュ３３に読み込まれる。ここで、トランザクション３５がメタデータの内容を更新すると、ログ採取手段３６によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ３７に保持される。そして、ログ書き込み手段３８により、ログバッファ３７内の情報がログボリューム３２に書き込まれる。その際、シーケンス番号採番手段３０ｂによりシーケンス番号が昇順で採番され、書き込むべきログに付与される。また、ファイルシステム復元手段３９により、ログボリューム３２に格納されたログを用いてメタボリューム３１内のメタデータの不整合が修正されると、修正した時に用いられたログの最後のシーケンス番号が初期シーケンス番号記憶手段３０ａに記憶される。その後、ログ書き込み手段３８により、ログバッファ３７内の情報がログボリューム３２に書き込まれると、シーケンス番号採番手段３０ｂにより、初期シーケンス番号記憶手段３０ａに記憶されたシーケンス番号を基準としてシーケンス番号が昇順で採番され、書き込むべきログに付与される。

【００５４】これにより、既にファイルシステムの復元に用いたログと、ファイルシステム復元後に介したトランザクションのログとのシーケンス番号が重ならないことを保証することができる。その結果、システムが使用可能な期間中にログボリュームをゼロクリアする必要がなくなり、ゼロクリアに伴う処理の遅延を避けることができる。

【００５５】図４は、第４の発明の原理構成図である。第４の発明は、同じメタデータに対して複数回更新処理が行われる場合に、最終形態のメタデータのみをログとして採取するものである。第４の発明は、以下のような要素で構成される。

【００５６】メタボリューム４１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。メタキャッシュ４２は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段４３は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ４２へと読み込む。トランザクション４４は、メタキャッシュ４２内に読み込まれたメタデータの内容を更新する。ログ採取手段４５は、トランザクション４４の種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクショ

ンの場合には、メタキャッシュ４２内で変更されたメタデータの最終形態のみをログとして採取する。ログバッファ４６は、ログ採取手段４５が採取した情報を保持する。

【００５７】このようなデータ処理装置によれば、トランザクション４４がメタデータの更新をする際には、まず、メタデータ読み込み手段４３によりメタボリューム４１内の必要なメタデータがメタキャッシュ４２に読み込まれる。ここで、トランザクション４４がメタデータの内容を更新する。すると、ログ採取手段４５により、トランザクション４４の種別が判断され、メタデータの更新を複数回行う可能性のあるメタデータ属性を予測する。予測されたメタデータ属性において、同一メタデータが何度も更新される可能性がある場合には、その属性のメタデータがメタキャッシュ４２内で変更された時点ではログ採取を行わず、トランザクション４４が終了した時点で、最終形態のメタデータをログとして採取する。採取した情報は、ログバッファ４６に保持される。

【００５８】これにより、複数回更新されたメタデータのログを更新処理の度に採取することがなくなり、メモリの効率化を図ることができるとともに、ログを書き出すときのＩ／Ｏの量も減らすことができる。

【００５９】図５は、第５の発明の原理構成図である。第５の発明は、メタデータ割り当て管理情報の一部の複製を生成し、複製として生成した情報内からのみメタデータの獲得を可能とし、解放する際には、割り当て管理情報においてのみ解放された旨の情報の更新を行うことで、解放直後に別のトランザクションに獲得されるのを防止したものである。第５の発明は、以下のような要素で構成される。

【００６０】割り当て管理情報保持手段５１は、メタデータに対する割り当てを管理するための割り当て管理情報５１ａを複数の領域に分割して保持する。獲得操作用管理情報生成手段５２は、割り当て管理情報保持手段５１内の割り当て管理情報の一部領域の複製を生成し、獲得操作用管理情報５１ｂとする。また、獲得操作用管理情報５１ｂ内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を獲得操作用管理情報５１ｂとする。トランザクション５３は、メタデータの獲得及び解放要求を出力する。メタデータ獲得手段５４は、トランザクション５３によりメタデータの獲得要求が出された場合には、獲得操作用管理情報５１ｂの中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように獲得操作用管理情報５１ｂと割り当て管理情報５１ａとの内容を変更する。メタデータ解放手段５５は、トランザクション５３によりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、割り当て管理情報の内容を変更する。

【００６１】このようなデータ処理装置によれば、獲得操作用管理情報生成手段５２により、割り当て管理情報５１ａの一部領域の複製が生成され、獲得操作用管理情報５１ｂとされる。トランザクション５３により獲得要求があると、メタデータ獲得手段５４により、獲得操作用管理情報５１ｂ内の未獲得のメタデータが獲得される。すると、獲得操作用管理情報５１ｂと割り当て管理情報５１ａとの内容が更新される。また、トランザクション５３よりメタデータの解放要求があると、メタデータ解放手段５５により該当するメタデータの解放処理が行われる。この際、割り当て管理情報５１ａの内容のみが更新される。ここで、獲得操作用管理情報５１ｂ内に未獲得のメタデータがなくなると、割り当て管理情報５１ａ内の別の領域の複製が獲得操作用管理情報５１ｂとされる。

【００６２】これにより、解放された旨の情報が獲得操作用管理情報５１ｂに反映されないため、解放直後のメタデータが他のトランザクションに獲得されることがなくなる。その結果、解放処理を行ったトランザクションの終了前にシステムがダウンしても、少なくとも解放前の状態のまま保全されることが保証される。

【００６３】図６は、第６の発明の原理構成図である。第６の発明は、トランザクション単位に、獲得や解放に関する情報をログとして記録することで、必要なメモリ容量の削減を図るとともに、平行動作するトランザクションのログに起因して割り当て管理情報に不正な状態が発生することを防ぐものである。第６の発明は、以下のような要素で構成される。

【００６４】割り当て管理情報保持手段６１は、メタデータに対する割り当てを管理するための割り当て管理情報を保持する。トランザクション６２は、メタデータの獲得及び解放要求を出力する。メタデータ獲得手段６３は、トランザクション６２によりメタデータの獲得要求が出された場合には、獲得操作用管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように割り当て管理情報６１ａの内容を変更する。メタデータ解放手段６４は、トランザクション６２によりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、割り当て管理情報６１ａの内容を変更する。ログ採取手段６５は、割り当て管理情報内のメタデータ獲得手段６３及びメタデータ解放手段６４によって変更された部分の情報をログとして採取する。ログバッファ６６は、ログ採取手段６５が採取したログを保持する。

【００６５】このようなデータ処理装置によれば、トランザクション６２からメタデータの獲得要求が出されると、メタデータ獲得手段６３によって未獲得のメタデータの１つが割り当て管理情報６１ａ内から獲得され、そのメタデータが獲得済みの状態とされる。また、トランザクション６２からメタデータの解放要求が出されると、メタデータ解放手段６４によって該当するメタデー

タが未獲得の状態に変更される。そして、ログ採取手段６５により、割り当て管理情報６１ａ内のメタデータ獲得手段６３及びメタデータ解放手段６４によって変更された部分の情報がログとして採取され、ログバッファ６６に保持される。

【００６６】このように、獲得、解放のログを採取する際に、トランザクションが獲得や解放を行ったという情報のみをログとして格納することで、メモリ等の領域を効率よく利用することができるとともに、他トランザクションによる獲得や解放処理の情報がログに含まれないことによって、割り当て管理情報が不正な状態となることを防ぐことができる。

【００６７】図７は、第７の発明の原理構成図である。第７の発明は、ログバッファを複数設けることにより、トランザクションの独立性をいっそう高めたものである。第７の発明の構成要素は以下の通りである。

【００６８】メタボリューム７１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。メタキャッシュ７２は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段７３は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ７２へと読み込む。複数のトランザクション７４ａ～７４ｃは、メタキャッシュ７２内に読み込まれたメタデータの内容を更新する。複数のログバッファ７５ａ～７５ｅは、ログをトランザクション毎に保持する。各ログバッファ７５ａ～７５ｅのサイズは一定ではなく、大きなサイズや小さなサイズが存在する。ログ採取手段７６は、メタキャッシュ７２内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて適したサイズのログバッファ７５ａ～７５ｅに格納する。例えば、最初にログを格納する場合には、トランザクションの内容によって予想される処理に適した大きさのログバッファに格納し、格納対象となるログバッファの記憶容量が不足してきたら、より大きな記憶容量のログバッファへログを移し替え、以後、より大きな記憶容量のログバッファをログの格納対象とする。

【００６９】このようなデータ処理装置によれば、トランザクション７４ａ～７４ｃがメタデータの更新をする際には、まず、メタデータ読み込み手段７３によりメタボリューム７１内の必要なメタデータがメタキャッシュ７２に読み込まれる。ここで、トランザクション７４ａ～７４ｃがメタデータの内容を更新する。すると、ログ採取手段７６により、メタキャッシュ７２内で変更されたメタデータがログとして採取され、適したサイズのログバッファ７５ａ～７５ｅに保持される。

【００７０】このように、複数のログバッファに分け、トランザクション毎に１つのログバッファを使用するようにしたことで、トランザクションの独立性を保つことができる。しかも、複数のサイズのログバッファを用意し、トランザクション毎に適したサイズのログバッファを使用することにより、メモリを効率的に利用できる。

【００７１】図８は、第８の発明の原理構成図である。第８の発明は、あるトランザクションのログがログバッファに入りきらない場合に、中間ログとしてログバッファの内容を書き出すものである。第８の発明の構成は以下の通りである。

【００７２】メタボリューム８１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム８２は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ８３は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段８４は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ８３へと読み込む。トランザクション８５は、メタキャッシュ８３内に読み込まれたメタデータの内容を更新する。ログバッファ８６は、ログをトランザクション毎に保持する。ログ採取手段８７は、メタキャッシュ８３内で変更されたメタデータの内容をログとして採取し、ログバッファ８６に格納する。ログ書き込み手段８８は、トランザクション８５が終了した場合にログバッファ８６の内容をログボリューム８２に書き込むとともに、トランザクション８５によるログがログバッファ８６内に格納しきれない場合には、ログバッファ８６内のデータを完結したログに加工し、中間ログとしてログボリューム８２内に格納する。なお、中間ログを生成する際には、中間ログに対してトランザクションを実行するのに必要とされたパラメタに関する情報を付加する。ファイルシステム復元手段８９は、ファイルシステム復元要求を受け取ると、ログボリューム８２に格納されたログを用いて、メタボリューム８１内のメタデータの不整合を修正する。このとき、中間ログを発見すると、中間ログに含まれたパラメタを用いてトランザクションを再実行させる。

【００７３】このようなデータ処理装置によれば、トランザクション８５がメタデータの更新をする際には、まず、メタデータ読み込み手段８４によりメタボリューム８１内の必要なメタデータがメタキャッシュ８３に読み込まれる。ここで、トランザクション８５がメタデータの内容を更新すると、ログ採取手段８７によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ８６に保持される。そして、ログバッファ８６に格納しきれなくなるか、トランザクション８５が終了すると、ログ書き込み手段８８により、ログバッファ８６内の情報がログボリューム８２に書き込まれる。ログバッファ８６に格納しきれなくなった場合には、ログバッファ８６の内容を完結したログに加工し、中間ログとしてログボリューム８２に格納する。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段８９により、ログボリューム８２に格

納されたログを用いて、メタボリューム８１内のメタデータの不整合が修正されるとともに、中間ログまで採取した段階で停止しているトランザクションが再実行される。

【００７４】これにより、メタデータの更新を大量に行うトランザクションの実行中にシステムがダウンした場合には、途中の状態まで戻すことができるとともに、トランザクションが再実行されることで、トランザクションの実行後の状態へ遷移させることができる。

【００７５】図９は、第９の発明の原理構成図である。トランザクションの受け入れを一定の条件によって制限することで、メモリ枯渇等を防止するものである。第９の発明の構成は以下の通りである。

【００７６】メタボリューム９１は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム９２は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ９４は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段９３は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ９４へと読み込む。互いの同時実行可能な複数のトランザクション９０ｂ～９０ｄは、メタキャッシュ９４内に読み込まれたメタデータの内容を更新する。トランザクション受け入れ制限手段９０ａは、トランザクション９０ｂ～９０ｄからの開始要求を受け付けると、ログ採取に関するシステムの動作状況に基づいて、トランザクション９０ｂ～９０ｄの受け入れ許否を判断する。受け入れ判断基準としては、例えばログボリューム内の有効なログが占める割合を用いる。すなわち、有効範囲監視手段９９によって有効なログとされたログがログボリューム中に占める割合が一定値以上である間は、トランザクション９０ｂ～９０ｄの受け入れを拒絶する。

【００７７】ログ採取手段９６は、メタキャッシュ９４内で変更されたメタデータの内容をログとして採取する。ログバッファ９５は、ログ採取手段９６が採取した情報を保持する。ログ書き込み手段９７は、ログバッファ９５が保持する情報を、適宜ログボリューム９２に格納する。メタデータ書き込み手段９８は、メタキャッシュ９４内のメタデータをメタボリューム９１内に格納する。有効範囲監視手段９９は、メタデータ書き込み手段９８による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応するログボリューム９２内のログを、有効なログとして指定する。

【００７８】このようなデータ処理装置によれば、トランザクション９０ｂ～９０ｄから開始要求が出されると、トランザクション受け入れ制限手段９０ａが受け入れの許否を判断する。例えば、有効範囲監視手段９９により有効であると指定されたログのログボリューム９２内に示す割合が一定以上の場合には、それ以上ログが発生しないように、トランザクションの受け入れを拒絶する。その後、メタデータ書き込み手段９８によるメタデータの書き込みが進み、有効なログの割合が減少したら、その時点でトランザクションの開始要求を許可する。

【００７９】これにより、ログボリュームの空き容量の減少に伴うハングアップなどの障害の発生を防止することができる。次に、本発明の実施の形態を具体的に説明する。

【００８０】図１０は、本発明を適用するデータ処理装置のハードウェア構成図である。データ処理システムは、ＣＰＵ(Central Processing Unit)２１１を中心に構成されている。ＣＰＵ２１１は、バス２１７を介して他の機器を制御するとともに、様々なデータ処理を行う。バス２１７には、メモリ２１２、入力機器インタフェース２１３、表示制御回路２１４、ＨＤＤ(Hard Disk Drive)インタフェース２１５、及びネットワークインタフェース２１６が接続されている。

【００８１】メモリ２１２は、ＣＰＵ２１１が実行すべきプログラムや、プログラムの実行に必要な各種データを一時的に保持する。入力機器インタフェース２１３は、入力機器としてキーボード２２１とマウス２２２が接続されており、これらの入力機器からの入力内容をＣＰＵ２１１に伝える。

【００８２】表示制御回路２１４は、表示装置２２３が接続されており、ＣＰＵ２１１から送られてきた画像データを表示装置２２３で表示可能な画像情報に変換し、表示装置２２３の画面に表示させる。

【００８３】ＨＤＤインタフェース２１５は、複数のＨＤＤ２３１～２３３が接続されており、ＣＰＵ２１１から送られてきたデータをＨＤＤ２３１～２３３に格納するとともに、ＣＰＵ２１１からの要求に応じてＨＤＤ２３１～２３３内のデータを読み取り、ＣＰＵ２１１に転送する。

【００８４】ネットワークインタフェース２１６は、ＬＡＮ(Local Area Network)に接続されており、ＬＡＮを介してデータ通信を行う。すなわち、ＣＰＵ２１１から送られたデータをＬＡＮに接続された他のコンピュータに転送するとともに、他のコンピュータからＬＡＮを介して送られてきたデータをＣＰＵ２１１に転送する。

【００８５】ＨＤＤ２３１～２３３には、各種ファイルや、そのファイルを管理するためのメタデータ及びログが格納されている。このような構成のシステムにおいて、ＣＰＵ２１１がＨＤＤ２３１～２３３に格納されたオペレーティングシステム用のプログラムを実行することにより、本発明のログ採取機能が実現される。

【００８６】図１１は、ファイルシステム上で動作するログ採取機能の構成図である。図のように、ファイルを管理するためのメタボリューム１１１～１１３も複数設

けられている。メタボリューム１１１～１１３には、それぞれメタデータが格納されている。メタデータは、ファイルの格納場所等を管理するために必要な情報を有している。

【００８７】ログボリューム１２０は、ログ１２２を格納するための二次記憶装置である。ログボリューム１２０には、ログ１２２の他にボリューム管理情報１２１が格納されている。

【００８８】メタキャッシュ１３０は、メタデータを操作するためのメモリ上の領域である。メタキャッシュ１３０内には、操作対象となるメタデータ１３２とそのメタデータの割り当て管理情報１３１とが格納される。

【００８９】ログキャッシュ１４０は、複数のログバッファ１４１～１４４を有している。ログバッファ１４１～１４４のサイズは均一ではなく、大きなサイズのものや小さなサイズのものがある。これらのログバッファ１４１～１４４には、メタキャッシュ１３０内で更新されたメタデータの複製がログとして格納される。

【００９０】ログキャッシュ１４０とは別にログライトバッファ１５０が設けられている。ログライトバッファ１５０には、トランザクションの処理が終了した時点で、ログバッファ１４１～１４４内のログが転送される。

【００９１】この例では、複数のトランザクション１０１～１０３が並列動作している。このトランザクション１０１～１０３は、ファイルシステムオペレーションを分割したものである。

【００９２】実際にＩ／Ｏを行うのは２つのデーモンであり、それぞれをメタライトデーモン１０４、ログライトデーモン１０５と呼ぶ。ログライトデーモン１０５がログをログ専用の二次記憶装置であるログボリューム１２０に出力する。メタライトデーモン１０４は、ログボリューム１２０に対してログが出力されたことを確認した後、そのログに対応するメタデータをメタデータ専用の二次記憶装置であるメタボリューム１１１～１１３に出力する。

【００９３】さらに、本発明のログ採取機能では、以下のような特徴を有している。第１の特徴は、各メタデータに対応するメタデータ管理情報として、メタボリュームを識別する情報が付加されていることである。これは、大規模ファイルシステムに対応するためのものである。すなわち、複数の二次記憶装置がメタボリューム１１１～１１３として定義される場合に、メタキャッシュ１３０上のメタデータ管理情報にそのメタデータが存在するボリュームの情報を持たせている。

【００９４】図１２は、メタデータ管理情報を示す図である。メタデータ管理情報には、「ボリューム番号」、「メタデータ番号」、及び「メタデータポインタ」が登録されている。「ボリューム番号」には、対応するメタデータが存在するボリューム番号が登録されている。「メタデータ番号」には、ボリューム毎におけるメタデータ管理番号が登録されている。すなわち、システムが認識するボリュームのデバイス番号とそのボリューム内の位置から算定される数値によってメタデータが管理され、メタデータ自体がそれらの値を管理情報として保持する。「メタデータポインタ」には、メタデータの実体のある場所を指し示している。

【００９５】このようなメタデータ管理情報を有するメタデータの内容が更新されると、更新後のメタデータの内容がログとしてログバッファに記録されるとともに、メタデータ管理情報の内容がログに記録される。

【００９６】図１３は、ログバッファの形式を示す図である。ログバッファには、「ＢＥＧＩＮマーク」、「ボリューム番号」、「メタデータ番号」、「メタデータ実体」、及び「ＥＮＤマーク」の情報を含んでいる。

【００９７】ファイルシステムを復元する際には、ログ内に記されているボリューム番号によって、そのログによって復元すべきメタデータの存在するメタボリュームが認識される。これにより、従来技術のようにファイルシステム復元時にメタデータ番号からその存在すべきボリュームを決定するよりも速度向上が望め、システムダウン後の大規模ファイルシステムにおいてもログ機構導入によるファイルシステム復元時間の短縮が有効に機能する。

【００９８】第２の特徴は、メタキャッシュ１３０内で更新されたメタデータが、それぞれの管理構造にリンクポインタを持つことである。ログとしてログボリューム１２０に記録されたメタデータはメタライトリストに繋がれ、メタボリュームへ反映が完了した時点でこのメタライトリストから外される。さらに、ログとして記録されているログボリューム１２０内の位置情報をも、メタライトリスト内に持つ。このログボリューム内位置情報をリストを辿って検索することによって、システムダウン時に必要とされるログの範囲を特定することができる。そこで、ここから得られる有効範囲情報をログボリューム１２０の特定位置に設けたボリューム管理情報１２１内に記録する。有効範囲情報を記録し、ファイルシステム復元時にそこに含まれるログのみをリプレイすることにより、ログボリューム全体を検索する必要がなくなり、ログボリューム全体を読み込む必要のある、有効範囲情報を記録しない従来の方式よりもファイルシステムの復元時間をさらに短縮することができる。

【００９９】ところで、ログ機構はシーケンシャル性を持ったディスクアクセスを行うことで、記録すべき内容をヘッドシークすることなしにディスクへ保存でき、ディスクアクセス時間の短縮を図っている。ところが、本手法を採用した場合、メタデータのメタボリューム反映時、ログボリュームへの書き出し時に、ログボリュームの特定位置に有効範囲情報を書き出すこととなる。これではシーク削減の意図が全く意味を成さない。

【０１００】そのため、本実施の形態ではある程度のインターバルを空けて、有効範囲情報は書き出すように工夫した。変更がある度に、常に書き込まれるわけではないため有効範囲情報として保存されている情報には若干の誤差が含まれてしまう。ファイルシステム復元時にその誤差を吸収する必要がある。

【０１０１】図１４は、有効範囲を説明する図である。図中において、「●」で示すのが、まだメタボリュームに反映が終了しておらず、ファイルシステム復元に必要なログを意味する。「○」で示すのは、メタボリュームに反映が完了したことによって、ファイルシステム復元時には利用しなくても良いログである。

【０１０２】従来のファイルシステムにおいて、ログを用いたファイルシステム復元では、ログボリューム全体を検索し，ログに記録されたシーケンス番号から、最古のログを求め、たとえそれがメタボリュームに反映済みの、利用しなくても良いログであっても利用して、ログボリューム全体のログを用いていた。

【０１０３】本発明では、ある時点で有効範囲情報を書き出した時の有効範囲が示されている。その後、有効範囲を書き出さずに、メタボリュームへの反映が進み、また、別のトランザクションのログがログボリュームに書き出されたことによって、実際の有効範囲と有効範囲情報とはズレを生じている。この時点でシステムがダウンし、ログを用いてファイルシステムを復元する場合には、多少のムダが生じるが、有効範囲情報が示す先頭位置から、ログを利用する。利用すべきログの末端は有効範囲情報以降の一定範囲を検索しなければならないが、その検索は有効範囲情報を書き込むインターバルに依存し、範囲が限られている。

【０１０４】第３の特徴は、ログに対してトランザクション終了時にシーケンス番号を与え、その番号にはデータサイズが肥大化することを考慮に入れた上で、十分大きいデータ型を適用することである。データ型の大きさは、コンピュータ自身の使用可能年数の間使い続けても枯渇しない程度のサイズとする。例えば、システムの年表記を４桁の十進数「最大９９９９」で表していた場合、西暦１万年まで使用されることは想定されていない。その場合、西暦９９９９年まで使用可能なデータ型とすれば、ログに対するシーケンス番号がオーバーフローして逆転することがないことを保証することができ、それにより、ログボリューム全体をゼロクリアして初期化する必要が生じない。具体的には、データ型を６４ビット型とすれば、オーバーフローすることは現実にはありえない（４万年ほど耐えられるものと思われる）。ファイルシステム管理情報、各トランザクションのログ、有効範囲情報にこのシーケンス番号を含め、ログボリュームに記録する。このように、ログに与えるシーケンス番号のデータ型に十分大きいものを適用することにより、通常の運用時にトランザクションが動作する毎にインクリメントしても、オーバーフローは現実には起こり得ない。

【０１０５】さらに、スーパブロックと呼ばれるファイルシステム全体の管理情報にこのシーケンス番号を含め、正常なアンマウント処理時及びファイルシステム復元時にスーパブロックに含まれるシーケンス番号を正しく設定し、次回のマウント時にそこから得られる値を用いる。これにより、シーケンス番号は必ず昇順となることが保証され、ファイルシステム復元時に新しいログを古いものと取り違えないことが保証される。

【０１０６】シーケンス番号の順序性を保証することによって、ファイルシステム復元時にログボリュームをゼロクリアしなくとも、常に正しいログを利用することが可能となり、ファイルシステム復元時にログボリューム全体をゼロクリアするのに比べ、大幅な時間短縮が可能となる。

【０１０７】以上の理由により、ログの最大のメリットであるファイルシステム復元の時間短縮がより一層有効に機能することが可能となる。第４の特徴は、トランザクションによって更新されたメタデータが、そのメタデータの属性に応じ、再度同一のトランザクションにおいて更新される可能性のあるメタデータであった場合には、更新の時点ではログバッファにはコピーせず、リスト構造（トランスリスト）によって管理することである。

【０１０８】そして、同一トランザクション内で複数回更新されるメタデータとして、領域割当て管理情報をリスト構造で管理し、トランザクション進行中にはその中途段階のログは採取しない。トランザクション終了時にリスト構造を辿って、トランザクションの更新の最終状態のみを一括してログとすることによって、ログの縮小が図られ、それに伴いファイルシステム復元時間の短縮が可能となる。

【０１０９】具体的には、メタキャッシュ１３０内のメタデータを管理する構造にトランスリストへのリンクポインタを持たせることによって実現している。トランザクションによって更新されたメタデータは、ただ一回しか更新されないことが分かっているメタデータである場合には、その時点でログバッファへコピーされるが、以降もトランザクション進行中に更新される可能性がある場合には、このリンクポインタを用いてリスト構造に繋がれる。メタデータの更新可能性の有無は、トランザクションの種別で判断する。例えば、データ領域の空きなどを管理するためのトランザクションでは、何度もメタデータが更新される。

【０１１０】そして、トランザクションが終了する時にこのトランスリストを辿り、メタデータの最終形態をログバッファへコピーする。すると、結局はトランザクションが同一メタデータを複数回更新しても、最終形態の一回だけのログ採取で済まされる。

【０１１１】図１５は、ログ採取処理のフローチャートである。この処理は、オペレーティングシステムを実行するＣＰＵが行う処理である。以下、ＣＰＵがオペレーティングシステムを実行することにより実現する機能を、単に「システム」ということとする。

［Ｓ１］トランザクションの開始宣言を行う。

［Ｓ２］ログ採取要求を行う。

［Ｓ３］対象メタデータの更新可能性の有無を判断する。更新可能性があればステップＳ５に進み、そうでなければステップＳ４に進む。

［Ｓ４］ログバッファへメタデータをコピーし、ステップＳ２に進む。

［Ｓ５］対象メタデータはトランスリストに繋がれているか否かを判断する。トランスリストに繋がれていればステップＳ７に進み、そうでなければステップＳ６に進む。

［Ｓ６］メタデータ毎にトランスリストに繋ぐ。

［Ｓ７］トランザクションの処理が終了するか否かを判断する。終了するのであればステップＳ８に進み、そうでなければステップＳ２に進む。

［Ｓ８］トランザクション終了宣言を行う。

［Ｓ９］トランスリストを辿り、繋がれている最終形態のメタデータをそれぞれログバッファへコピーする。

【０１１２】このようにして、トランザクションの更新の最終状態のみを一括してログとして保存することができる。第５の特徴は、メタボリュームの割当て管理情報は獲得用として通常の割当て管理情報の複製を新たに設けたことである。ここで、割り当て管理情報として、ビットマップを例に挙げて説明する。

【０１１３】図１６は、メタボリュームの割り当て管理状況を示す図である。割り当て管理情報１３１には、使用されているメタデータを管理するためのビットマップ１３１ａが用意されている。ビットマップ１３１ａは複数のブロックに分けられている。そして各ブロックのビットマップの各ビットが「０」か「１」かによって、対応するメタデータが空いているか否かが示される。そして、ブロックに分けられたビットマップの１つの複製が作られ、獲得用ビットマップ１３１ｂとされる。

【０１１４】獲得時には、通常のビットマップ操作と同様に未割り当て状態の領域の検索が行われる。この時、検索対象として用いるのは獲得用ビットマップ１３１ｂである。獲得用ビットマップ１３１ｂ内から未割り当て状態の領域（ビットが立っていない）が見つかった時には、その獲得要求には見つかったビット番号を返す。そして、獲得用ビットマップ１３１ｂ及び、その複製元となったビットマップの対応ビットを立てる。一方、獲得用ビットマップ１３１ｂ内から未割り当て状態の領域が見つからなかった時には、別のブロックのビットマップの複製を生成し、新たな獲得用ビットマップ１３１ｃとする。

【０１１５】図１７は、ビットマップによるメタデータ獲得処理を示すフローチャートである。この処理は、システムが行う処理である。

［Ｓ１１］獲得要求を発行する。

［Ｓ１２］獲得用ビットマップに空きがあるか否かを判断する。空きがあればステップＳ２０に進み、そうでなければステップＳ１３に進む。

［Ｓ１３］メモリ（メタキャッシュ１３０）上のビットマップに「FreeDirty」フラグが立てられていないビットマップが存在するか否かを判断する。存在すればステップＳ１４に進み、存在しなければステップＳ１６に進む。ここで「FreeDirty」フラグとは、一回以上の解放処理が対象ビットマップに対してなされたことを意味する。

［Ｓ１４］「FreeDirty」フラグが立てられていないビットマップの中で、空きのあるものがあるか否かを判断する。そのようなビットマップがあればステップＳ１５に進み、そうでなければステップＳ１６に進む。

［Ｓ１５］「FreeDirty」フラグが立てられておらず、空きのあるビットマップの複製を、獲得用ビットマップに作成する。その後、ステップＳ２０に進む。

［Ｓ１６］メタボリューム１１１～１１３上のビットマップに空きがあるか否かを判断する。空きのあるビットマップがあればステップＳ１７に進み、そうでなければステップＳ１８に進む。

［Ｓ１７］メタボリューム１１１～１１３上の空きのビットマップをメモリ（メタキャッシュ１３０）上に読み込み、ステップＳ１５に進む。

［Ｓ１８］メモリ（メタキャッシュ１３０）上のビットマップに「FreeDirty」フラグが立てられたビットマップが存在するか否かを判断する。存在すればステップＳ１９に進み、存在しなければ、獲得不可能と判断し処理を終了する。

［Ｓ１９］「FreeDirty」フラグが立てられたビットマップをメタボリュームに反映し、「clean」状態にする。ここで「clean」状態とは、獲得、解放の処理が全く行われていない状態を示す。その後、ステップＳ１３に進む。

［Ｓ２０］獲得用ビットマップのビットを立てる。

［Ｓ２１］複製元ビットマップの同一ビットを立てる。

［Ｓ２２］複製元ビットマップに「AllocDirty」フラグを立てる。ここで、「AllocDirty」フラグとは、対象ビットマップから一回以上の獲得処理によって更新がなされた場合に、そのビットマップの管理構造体内のフラグに立てる値である。「AllocDirty」フラグや「FreeDirty」フラグが立ったビットマップはログ採取対象である。メタボリュームに反映された時にこれらのフラグは落とされ、「clean」状態となる。

【０１１６】このようにして、空きのビットマップを獲得できる。解放時にも、通常のビットマップ操作と同様

に、要求された領域が対応するビットの算出がまず行われるが、対応するビットを落とす操作は、対象のビットマップに対してのみ行い、仮にその対象ビットマップの複製が獲得用ビットマップとして存在する場合にも、その獲得用ビットマップに対しては行わない。

【０１１７】図１８は、解放処理のフローチャートである。この処理は、システムが行う。

［Ｓ３１］解放要求を出す。

［Ｓ３２］対象ビットマップがメモリ（メタキャッシュ１３０）上にあるか否かを判断する。対象ビットマップがあればステップＳ３４に進み、そうでなければステップＳ３３に進む。

［Ｓ３３］メタボリューム１１１～１１３上の対象ビットマップをメモリ（メタキャッシュ１３０）上に読み込む。

［Ｓ３４］対象ビットマップの対応するビットを落とす。

［Ｓ３５］対象ビットマップに「FreeDirty」フラグを立てる。

【０１１８】この一見複雑に見える作業によって、解放した領域を即座に別の用途に利用してしまうことを回避することが可能となり、システムダウン時に中途までしか終了していなかった解放トランザクションが解放したはずである領域は、解放される直前の状態のまま保全されることが保証できる。

【０１１９】例えば、Ａというトランザクションが解放処理、Ｂというトランザクションが獲得処理を行う場合を考える。ここで、トランザクションＡの解放処理はトランザクションＢの獲得処理よりも先に行われ、かつ、トランザクションＡはトランザクションＢよりも後に終わる場合を例に挙げる。

【０１２０】図１９は、トランザクションの処理の開始と終了の状況を示す図である。この図において、各トランザクションは「ＢＥＧＩＮ」で始まり、「ＥＮＤ」で終わることを意味し、トランザクションの「○」が解放処理を、「●」が獲得処理を意味する。

【０１２１】上図のように、トランザクションＡの解放処理がトランザクションＢの獲得処理よりも先に行われ、かつ、トランザクションＢのほうがトランザクションＡよりも先に終了した場合に従来のログ採取方式では問題が生ずることは既に述べた。

【０１２２】本発明が提案する手法を用いれば、トランザクションＢが獲得する領域がトランザクションＡが解放した領域となることは基本的にはなく、領域枯渇状態に近く、どうしてもこの領域を獲得せねばならない時には、一旦、該当ビットマップをメタボリュームに反映した後、獲得処理が行われる。これにより、ファイルシステム復元時に必要とされる、メタボリュームに未反映なメタデータを対象とするログを用いた復元では、管理情報上、二重獲得と見えるようなログは残り得ない。

【０１２３】また、トランザクションＢのログにはトランザクションＡの解放処理が記録されないため、トランザクションＢのログよりも後に復元に用いるトランザクションＡのログによって、トランザクションＢが獲得した領域を変更されることもありえない。

【０１２４】第６の特徴は、前述のようにログ機構の導入にあたり、獲得・解放操作のログとして、その獲得・解放した対象の情報（このビットを立てた・落した）のみを記録することである。これによって、複数の獲得・解放要求に対して、要求の発生後、トランザクションの終了時点までをシリアライズすることなくログ管理でき、かつログ採取量は減少し、複数トランザクション動作による変更をメタボリュームへ反映する回数をも減少させることができる。

【０１２５】第７の特徴は、ログキャッシュ１４０を複数のログバッファで管理し、この時、いくつかの異なるサイズとすることである。ログキャッシュ１４０を分割して複数のログバッファとして管理することによって、トランザクションの独立性を確立することが容易となるとともに、有限なメモリ空間を有効に活用することが可能となる。

【０１２６】まず、トランザクションがメタデータを更新するより以前に、自分がメタデータを更新する旨、システムに宣言する。この時、トランザクションの種別より予測されるログ採取情報に応じて、最適なサイズのログバッファがシステムより与えられ、トランザクション進行に伴い蓄積されるログはここへ溜められることとなる。上記第６の特徴で説明した手法に加え、トランザクション毎に異なるログバッファを用いることによって、複数のトランザクション更新の混じらない、純粋なログとして残すことが可能である。

【０１２７】第８の特徴は、上記第７の特徴で説明した手法に加え、トランザクションの開始時に予測したログ採取量よりも多くのログを採取しなければならなかった場合に、与えられたログバッファから、さらに大きいログバッファへ移行することである。システムの状況に応じて、トランザクションによって残されるログのサイズは変動し、その差異を複雑な処理を必要とせず、かつ、有限なメモリ空間を有効に活用して吸収することが可能である。

【０１２８】第９の特徴は、ログバッファに入りきらない場合には、ログボリュームへ中間ログを書き出すとともに、中間ログを書き出すことによる矛盾の発生を防止する機能を備えたことである。

【０１２９】ログボリュームに対するＩ／Ｏ発行を減少させるために、ログライトバッファと呼ぶ二次キャッシュ領域を設けている。終了したトランザクションのログは、ログバッファからこのログライトバッファへ移され、適時にログライトデーモン１０５によってログボリュームへ書き出される。この適時とは、負荷が低い場合

には一定の周期で良いが、トランザクションによるメタデータの更新量が多く、最大のログバッファでも収まらない場合などには強制的なＩ／Ｏ発行が必要とされる。このような場合において、中途の段階でのログの出力では、まだ更新されていない（かもしれない）メタデータを含めた書き出しを行うことがファイルシステムの整合性を保つためには必要である。

【０１３０】このように有限なメモリ空間を有効活用することを考慮した上で、ログバッファ内に収まりきらないログをトランザクション途中で出力する時に、残されるログによって復元されるファイルシステムの整合性を正当なものとするための手法を提案している。

【０１３１】ファイルシステムに対する負荷がそれほど高くない場合には、ログのログボリュームに対するＩ／Ｏを極力少なくするために事前に与えられた周期に応じて定期的にデーモンによって行われる。しかし、ログバッファが枯渇し、以降のトランザクション進行で溜められるログが収まらないと判断された時、部分的なログとして、この時点のログをログボリュームへ出力する。この時、まだ更新されていない情報をもログに出力する。例えばファイルを追加ライトするトランザクションの場合は、部分的なログを出力する時点でのファイルサイズや時刻情報を合わせてログに記録する。これによって、部分的であるはずのこのログを、ファイルシステム復元時には完結した１つのトランザクションのログと同等に扱うことが可能となり、ファイルシステム復元時に複雑な考慮の必要なしに、望ましい状態にファイルシステムを復元することが可能となる。

【０１３２】図２０は、ログバッファへのログの格納状況を示す図である。この例では、２種類のサイズのログバッファ１４１～１４６が設けられている。ログバッファ１４１～１４５は通常のサイズであり、ログバッファ１４６が大きなサイズである。

【０１３３】また、ログバッファ１４１～１４６を管理するためのログバッファ管理テーブル１４８，１４９が設けられている。ログバッファ管理テーブル１４８は、ログバッファ１４１～１４６に対応するフラグビットを有している。そして、使用されているログバッファに対応するフラグビットの値が「１」に設定され、未使用のログバッファに対応するフラグビットの値は「０」に設定されている。同様に、大きいサイズのログバッファ１４６に対応するログバッファ管理テーブル１４９もフラグビットを有しており、そのフラグビットの値によって、ログバッファ１４６が使用されているか否かを管理している。

【０１３４】ログキャッシュ１４０内の各ログバッファ１４１～１４５のうち、ログバッファ１４３とログバッファ１４５とが現在利用中であることを意味するために、「●▲」などの記号を用いた。ここで、ログバッファ１４５には多くのログが溜まっており、既に満杯の状態となっている。このように通常のサイズのログバッファ１４１～１４５では容量が足りなくなった際には、そのログを大きいサイズのログバッファ１４６に移動する。そして、トランザクションが継続している間は、更新されたメタデータの内容を、随時ログバッファに格納していく。ここで、トランザクションが終了したら、そのトランザクションに対応するログバッファの内容をログライトバッファ１５０へ転送する。

【０１３５】ログライトバッファ１５０には、複数のバッファ１５１，１５２が設けられており、それらのバッファ１５１，１５２にログが書き込まれる。ログライトバッファ１５０内のログは、ログライトデーモン１０５によって随時ログボリュームに書き込まれる。

【０１３６】図２１は、ログ採取手順を示すフローチャートである。これはシステムが行う処理である。

［Ｓ４１］BEGIN宣言を行う。

［Ｓ４２］ログバッファの予約をする。

［Ｓ４３］ログ採取要求を行う。

［Ｓ４４］現在のログバッファに今回のログが収まるか否かを判断する。収まるのであればステップＳ５１に進み、収まらないのであればステップＳ４５に進む。

［Ｓ４５］現在のログバッファより大きいログバッファが存在するか否かを判断する。存在すればステップＳ４６に進み、そうでなければステップＳ４７に進む。

［Ｓ４６］現在のバッファから大きいバッファへ内容をコピーし、ステップＳ４４に進む。

［Ｓ４７］トランザクションに与えられたパラメタをログ採取する。

［Ｓ４８］現時点のファイル状態を正しく表す管理情報をログ採取する。

［Ｓ４９］現在の中途段階のログをログライトデーモン１０５に書き出させる。

［Ｓ５０］現在のログバッファをクリアする。

［Ｓ５１］ログを採取する。

［Ｓ５２］ログ採取要求を終了する。

［Ｓ５３］ＥＮＤ宣言があるか否かを判断する。END宣言があればステップＳ５４に進み、そうでなければステップＳ４３に進む。

［Ｓ５４］END宣言を行い、処理を終了する。

【０１３７】第１０の特徴は、トランザクションが完了する前に、システムダウンが発生した場合に備え、分割して出力するログにはトランザクションに与えられたパラメタを含め、ファイルシステム復元時にそのパラメタを元に、再度トランザクションを実行することである。

【０１３８】さらに、この部分的なログにトランザクションに与えられたパラメタを記録し、ファイルシステム復元時にそれを用いて、通常のログだけでは中途で終わった状態までしか復元できないファイルを、トランザクションが終了した状態にまでファイルシステムに反映することが可能となる。

【０１３９】図２２は、ファイルシステム復元処理を示すフローチャートである。これはシステムが行う処理である。

［Ｓ６１］システムダウン等が発生する。

［Ｓ６２］ファイルシステムの異常を検知する。

【０１４０】以降、ファイルシステム復元処理を行う。

［Ｓ６３］ログ開始マークを検出する。

［Ｓ６４］ログ開始マークに対応するログ終了マークが存在するか否かを判断する。ログ終了マークが存在すればステップＳ６５に進み、ログ終了マークが存在しなければステップＳ６６に進む。

［Ｓ６５］開始マークから終了マークまでのログを用いて復元処理を実行する。その後、ステップＳ６３に進む。

［Ｓ６６］対象トランザクションが以前のログだけで完結するか否かを判断する。完結するのであれば復元処理を終了し、完結しないのであればステップＳ６７に進む。

［Ｓ６７］ログに記録されたパラメタを読み込む。

［Ｓ６８］トランザクションとして行いたかった処理を理解する。

［Ｓ６９］ファイルに対して直接処理を再実行する。その後、復元処理を終了する。

【０１４１】これにより、オペレーションのセマンティクスを保証したファイルシステムの復元が可能となり、従来技術では完全に元に戻す、または完全に再実行することが不可能であったトランザクションを、完全に再実行することによってファイルの整合性をも回復することが可能となる。

【０１４２】第１１の特徴は、メタデータを更新するトランザクションが更新処理を行う前に、その旨をシステムに対し宣言することである。さらに、システムは宣言を受け入れるか否かを判断する機構を持つことである。ここでは、以下の条件の場合には新規のトランザクションの受け入れを拒否し、拒否されたトランザクションは再度認可が下りるまで待合わせなければならない。

・メタキャッシュ１３０がフルに近い状態にある場合。

・トランザクションの多重度が、システムで規定された値を既に越えていた場合。

・ログボリュームに残されたログの大部分が有効なログ状態である場合。

【０１４３】これらの場合に、ファイルシステムが新規トランザクションの受け入れを拒否し、トランザクションの多重度を制限することによって、結果としてダーティなメタデータの数を制限することが可能となり、メタキャッシュ１３０の空き領域枯渇などを要因とするハングアップ状態に陥ることを回避することが可能となる。特に、分割ログとしてログ採取しなければならないトランザクションが動作中に、新規トランザクションの開始を拒否することは、システムの状態を正常に維持する上で効果が高い。

【０１４４】第１２の特徴は、新規トランザクションの受け入れ拒否の機構をログボリューム内の有効ログの割合に応じて適用することにより、単一の、多数のメタデータを更新するトランザクションのみならず複数のトランザクションが更新したメタデータによって空き領域の減少に伴うハングアップ状態をも回避することである。

【０１４５】以下に、第１１の特徴と第１２の特徴とを含むトランザクションの受け入れ処理について説明する。図２３は、新規トランザクションの受け入れ許否判定処理を示すフローチャートである。

［Ｓ７１］トランザクションを行うプロセス（以下、単にトランザクションという）が、トランザクション処理を開始する。

［Ｓ７２］「BEGIN」宣言を行う。この際、オペレーティングシステム（以下、単に「システム」という）に対して問い合わせを行う。

［Ｓ７３］トランザクションは、システムからの回答待ち状態となる。システムより「ＯＫ」の回答を受け取ったらステップＳ８２に進む。

［Ｓ７４］トランザクションからの問い合わせを受けたシステムが、パラメタの評価を開始する。

［Ｓ７５］システムは、既に多重度が規定値より上であるか否かを判断する。多重度が規定値より上であれば「ＮＧ」としてステップＳ８１に進み、そうでなければ「ＯＫ」としてステップＳ７６に進む。

［Ｓ７６］システムは、現在サブトランザクションが動作中であるか否かを判断する。サブトランザクションが動作中であれば「ＮＧ」としてステップＳ８１に進み、そうでなければ「ＯＫ」としてステップＳ７７に進む。ここでサブトランザクションとは、ログを複数に分割して格納する場合に、１つのログバッファにログを格納できるような処理単位に分割されたトランザクションである。

［Ｓ７７］システムは、メタキャッシュ１３０に十分な空きがあるか否かを判断する。十分な空きがなければ「ＮＧ」としてステップＳ７９に進み、十分な空きがあれば「ＯＫ」としてステップＳ７８に進む。

［Ｓ７８］システムは、ログボリュームに上書き可能領域が少ないか否かを判断する。上書き可能領域が十分になければ「ＮＧ」としてステップＳ７９に進み、十分な上書き可能領域があれば「ＯＫ」としてトランザクションに対して「ＯＫ」の回答を返す。

［Ｓ７９］システムは、ログライトデーモン１０５を起動する。

［Ｓ８０］システムは、メタライトデーモン１０４を起動する。

［Ｓ８１］システムは、ＮＧ条件が解消されるまでスリープ状態で待つ。ＮＧ条件が解消されたら、ステップＳ７５に進む。

［Ｓ８２］トランザクションは、システムからの「ＯＫ」の回答を受け取ったら、多重度インクリメント処理を行う。
［Ｓ８３］トランザクションは、「BEGIN」処理を終了する。
［Ｓ８４］トランザクションは、処理に応じて、メタデータをキャッシュに読み込み、その度に空き量をデクリメントする。分割ログ化するなら他トランザクションの開始を拒否するようにシステムに依頼する。
［Ｓ８５］トランザクションは、「END宣言」を行う。
［Ｓ８６］トランザクションは、多重度をデクリメントする。
［Ｓ８７］トランザクションは、必要に応じてログ採取を繰り返した後、自己のトランザクション処理を終了する。
【０１４６】次に、本発明を適用したシステムの具体的な処理内容について説明する。本発明の全ての特徴を備えたシステムにおけるログ採取手順は以下のようになる。ファイルシステムオペレーションを分割したトランザクションはメタデータを更新するより前に、自分がこれからメタデータを更新する旨を宣言する（ＢＥＧＩＮ）。ＢＥＧＩＮ時にトランザクションに対し、独立したログバッファが割当てられる。この時、既にトランザクションの並列動作数が非常に多かったりメタキャッシュ１３０域の利用率が高い場合には、システムによってＢＥＧＩＮ宣言が拒否される。ＢＥＧＩＮ宣言を拒否されたトランザクションはその拒否理由が解消されるまで、待ち合わせなければならない。
【０１４７】更新が完了したメタデータは、ＢＥＧＩＮ時に割り当てられたログバッファへコピーされる。それと同時にメタデータ種類に応じたリストへ繋がれる。更新すべき全てのメタデータを更新し終わったトランザクションは、そこで完了を宣言する（ＥＮＤ）。ＥＮＤ時には、これまで溜めたログバッファの内容をログ専用の二次キャッシュであるログライトバッファへ移動し、さらに、まだログバッファへコピーされていなかったメタデータをログライトバッファへコピーする。
【０１４８】また、ＥＮＤ時にメタデータの種類に応じて繋がれていたリストから、そのトランザクションが更新した全てのメタデータを「ログ待ちリスト」へ繋ぎ替える。
【０１４９】以上で非同期要求のトランザクションは終了することができる。トランザクションが同期要求であった場合には、ログを書き出すまで待ち合わせなければならない。
【０１５０】ログの書き出しは独立したデーモン（ログライトデーモン１０５）が行う。このデーモンの起動契機は、同期要求トランザクションによる起動要求（ｗａｋｅｕｐ）、メタキャッシュ１３０の空き状態監視機構による起動要求（ｗａｋｅｕｐ）、及びタイマである。
【０１５１】起動したログライトデーモン１０５は複数のトランザクションのログがまとめられたログライトバッファを１つのＩ／Ｏとして発行する。そして、ログ待ちリストから「書き出しリスト」へメタデータを移動する。
【０１５２】メタボリュームへＩ／Ｏを発行するのは、メタライトデーモン１０４の役割である。メタライトデーモン１０４は書き出しリストに繋がれたメタデータを順に非同期ライトによってメタボリュームに反映する。メタライトデーモン１０４の起動契機は、ログバッファの空きが少なくなった時、ログボリュームの空きが少なくなった時、及びタイマである。
【０１５３】以上が、メタデータを更新するトランザクションの開始から、更新されたメタデータがメタボリュームに反映されるまでの大まかな流れである。以降に、ログの採取とそのディスク反映手法、そして、メタデータのディスク反映の処理について詳細に説明する。
（１）ログの構造
（１．１）ログボリューム
ログボリュームに格納される情報は、以下のような情報である。
【０１５４】ログボリュームにはスーパブロック、ボリューム管理情報が先頭にある。その次にログの有効範囲を示す構造体を記録する。構造体は以下のメンバを持つ。
・有効ログ先頭のログボリューム内オフセット
・有効ログ先頭のログシーケンス番号
・有効ログ末尾のログボリューム内オフセット
・有効ログ末尾のログシーケンス番号
ただし注意しなければならないのは、ここで先頭・末尾と言っているのは、真の値ではない可能性があることをリプレイ時に考慮しなければならない点である。なぜなら、ログはボリュームをシーケンシャルアクセスすることによってシーク時間の短縮を計っているが、このようにボリュームの一部へ毎回アクセスしたのでは、そのシーケンシャル性が損なわれてしまう。そのために、ログの書き出し毎にはこの有効範囲の書き出しは行わず、数回に一回、書き出している。
【０１５５】これにより、上記構造体に収められたオフセットは正確ではないためにリプレイ時に検索が必要ではある。しかしながら、全体を検索するよりも大幅に検索量が減るため、利点は保持できるものと考える。
【０１５６】上記以外は、全て、メタデータの更新履歴によって構成される。
（１．２）ログブロックの基本構造
ログボリューム１２０内に残されたログ（メタデータの更新情報）は基本的にはトランザクション単位である。これをログブロックと呼ぶ。しかし、キャッシュ域フルなどによって、１つのトランザクションを一度にまとめることができない場合は複数の分割ログに分ける。この

分割ログをサブトランザクションログと呼ぶ。

【０１５７】サブトランザクションログをリプレイすることによって、ファイルシステムの整合性が保てるよう、その内容は工夫されている。しかしながら、トランザクションの途中までのサブトランザクションをリプレイしたのでは、トランザクション全体が終わっていないことから、ファイルシステムとしての整合性が保てても、ファイルの中身は異常であるという状態に陥ってしまうことが考えられる。

【０１５８】この状態を回避するために、そのサブトランザクションに別れたトランザクションがどのようなオペレーションであったかをログに採取する（オペレーションログ）。ログリプレイ時には、サブトランザクションをリプレイした後、オペレーションログをリプレイヤが再実行することによって、トランザクション全体が終わった状態とすることが可能である。

【０１５９】サブトランザクション化する可能性のあるトランザクションは全体終了時に「終わったこと」をログに記録し（最終ＥＮＤマーク）、リプレイヤはその有無を調べてオペレーションログの実行を判断する。

（１．３）ログの採取形式

メタデータの更新情報は該当するメタデータの管理構造単位で採取する。また、メタボリュームの管理構造であるビットマップはその更新部分だけのログ採取とする、スーパブロックについては特殊であり、データ空き容量についてのみログ採取を行う。

（１．３．１）ｉｎｏｄｅ、Ｖデータ、空き管理情報

ファイルの管理情報であるｉノード、Ｖデータと呼ぶディレクトリやシンボリックリンクデータや空き管理情報のメタデータ本体は、それらのＩ／Ｏ単位（ブロック単位）でのログ採取を行う。

【０１６０】これは、ファイルシステム整合性チェック処理であるｆｓｃｋによるログリプレイ時の処理を簡略化することを意識している。変更された部分だけをログ採取した場合、ログリプレイ時には、該当ブロックの算定→読み込み→変更部分の更新→再度書き込み、とステップが増えてしまう。しかし、ブロック全体のログ採取によって、リプレイ時にはメタボリュームにログ情報を上書きするだけでよい。

【０１６１】ｉノード本体やディレクトリブロックなどのＶデータは、更新中はファイルのロックで保護されている。しかしながら、空き管理情報についてはファイルのロックでは保護されない。そのため、トランザクション途中で他トランザクションによって更新され、そのトランザクションが追い越して先に終わってしまうと、ログには古い情報が後に残されてしまい、リプレイするとファイルシステムに不整合が生じてしまう。

【０１６２】そのため、空き管理部については、ここでは更新するトランザクションが並行動作しないことを保証することによって実質的な排他制御を行い、他のメタデータと同じ採取形式とした。

（１．３．２）ビットマップ

メタデータの割り当て状況を管理するビットマップも空き管理部と同様にファイルのロックで保護されていない。そのため、ビットマップ全体のログ採取を行うと、そのログには複数のトランザクションによる更新が含まれてしまう可能性がある。特に、トランザクションの追い越しが発生すると、新しいはずのログによって、ビットマップの情報が古く書き戻されてしまうことが考えられる。

【０１６３】そこで、ビットマップのログ採取は更新部分だけとする。更新部分とは、あるビットが０になった、あるいは１になったと記録するだけである。それにより、トランザクションが並行動作してもそれぞれのトランザクションが更新した内容のみがログに残されるため、メタデータの後退は起こり得ない。

【０１６４】注意すべきは、ある領域を解放（「１→０」）した後、他トランザクションがそこを獲得（「０→１」）した場合である。トランザクションの追い越しが発生すると、ログリプレイによって獲得した領域を解放してしまう。

【０１６５】これについては、アロケート用ビットマップを１つ定め、複製を用いて操作を行うことによって回避する。

（１．４）ログの記録構造

（１．４．１）一般ログ

ログ有効状態であっても、ある程度の複数スレッドが同時に進行することを許す。これは性能要件より必須である。しかし、ログボリューム内のログは前述の通り、トランザクション毎に保存する。１つのトランザクション結果をログに記録する際、ログは以下のパーツで構成する。

１）ＢＥＧＩＮマーク
２）ヘッダ
３）更新内容
（２）＋３）の繰り返し）
４）ＥＮＤマーク

これを以下、ログブロックと呼ぶ。ログブロックはログボリュームの物理ブロック境界から始まるが、その終わりは物理ブロック境界であるとは限らない。

１）ＢＥＧＩＮマーク

トランザクションの開始時に作成される情報であり、以下の内容を含む。

・マジックワード
・トランザクションタイプ
・ログシーケンス番号
・ログブロックサイズ

マジックワード；１つのトランザクションが始まったことを示すマジックワードを埋め込む。

【０１６６】トランザクションタイプ：ＢＥＧＩＮマー

クにトランザクションのタイプ（種類）を埋め込むことによって、その後ろの更新情報に含まれる内容の概要を事前に知ることが可能。

【０１６７】ログシーケンス番号：ログに記録されたトランザクション毎にインクリメントされる数値。このカウンタが最小のログブロックがそのログボリューム内で最も古いトランザクションであることを示す。カウンタは６４ビット型とし、オーバーフローすることは現実にはありえない（４万年ほど耐えられるものと思われる）。ログリプレイ時に、ログボリュームをゼロクリアすることにより、再度０から開始することも考えられるが、リプレイ時間の短縮を考慮し、利用したログの最後のシーケンス番号より昇順に採番することによりゼロクリアを回避する。

【０１６８】ログブロックサイズ：ログブロックのＥＮＤマークまでのサイズを記録する。ＢＥＧＩＮマークの先頭からＥＮＤマークの先頭までのサイズである。リプレイ時にはＢＥＧＩＮマークの先頭からこのサイズだけ移動し、ＥＮＤマークの情報を参照して、ログの有効性を判定する。

２）ヘッダ

更新されたメタデータについて、その保管位置を特定するための情報であり、以下のメンバによって構成される。

・メタデータタイプ

・メタボリューム番号

・ボリュームローカルメタデータ番号

メタデータタイプ：更新情報の後方にあるメタデータ更新内容が何のメタデータであるのかを特定するために用いられ、リプレイヤはここから更新内容のサイズを判断する。

【０１６９】メタボリューム番号：メタデータ管理で用いているメタボリューム毎の番号を記録する。リプレイ時には、この番号から書き戻すメタボリュームを決定する。

ボリュームローカルメタデータ番号：メタデータ管理で用いている、メタボリューム毎、メタデータ毎に０から始まる値を記録する。リプレイ時には、この番号からメタボリューム内のブロック位置に変換し、更新内容を書き戻す。これはブロック位置変換を削減することによるログ採取の高速化が狙いである。対象がビットマップである場合には、変更されたビットが該当するメタデータ本体のメタデータ番号を記載する（ビットマップ番号ではない）。

３）更新内容

メタデータ本体の場合には、更新内容が含まれるブロック（それぞれのメタデータの管理単位）である。例えば、ディレクトリブロックが更新された場合には１０２４バイトのディレクトリブロック全体である。

【０１７０】ビットマップの場合には、全体ではなく、ここには「０→１」または「１→０」という情報だけ記録する。リプレイ時にはヘッダから該当するビットマップを判定し、それを読み込み、ここの内容から該当ビットを変更して書き戻す。

４）ＥＮＤマーク

ＢＥＧＩＮマークに対応するマジックワード、ログシーケンス番号、ログブロックサイズを記録する。

【０１７１】ＥＮＤマークがマジックワードとログ番号だけでは、メタデータの内容によってはそれが偽りのＥＮＤマークとして見えてしまい、リプレイの時に処理を誤る可能性がある。これを回避するために、ＥＮＤマークは固有形式（メタデータを誤認することのない形式）としなければならない。

【０１７２】ＢＥＧＩＮマークを固有形式としないのは、ＢＥＧＩＮマークだけ書き出された時点で、システムダウンした場合を考慮し、いずれにせよＥＮＤマークの識別ができなければならないからである。

【０１７３】固有形式とするために、固有の数値を６４バイト分続ける。これにより、全てのメタデータがＥＮＤマークに化けることはなくなる。これに続けて、マジックワード、カウンタ（ログ番号）、ログブロックサイズを記録する。以降、ブロック境界まで、上記数値を埋める。ＢＥＧＩＮマークに対応したＥＮＤマークを見つけることができないログ情報はリプレイしてはならない（ログ書き出し途中にシステムが異常終了したことを意味する）。

（１．４．２）巨大トランザクションのログ

トランザクションが多くのメタデータを更新する場合には、ログバッファサイズ、ログボリュームサイズが有限であることから、複数のサブトランザクションログに分割する。サブトランザクションログはそれだけのリプレイによって、ファイルシステムの整合性は保たれる。しかし、トランザクション全体のサブトランザクションログをリプレイしなければ、ファイルの整合性が保てない。そのため、サブトランザクション途中でのシステムダウンを考慮し、オペレーションログを内部に含む。

１）ＢＥＧＩＮマーク

２）オペレーションログ

３）ヘッダ

４）更新内容

（３）＋４）の繰り返し）

５）ＥＮＤマーク

（１）～５）の繰り返し）

（５’）最終ＥＮＤマーク）

１）ＢＥＧＩＮマーク

一般ログのＢＥＧＩＮマークと同じ。ただし、サブトランザクションログとなりうるトランザクションは、書き込みや削除など、データ領域を触るものや、領域管理に関わるものに限定される。

【０１７４】これらのトランザクションがＢＥＧＩＮマ

ーク内で設定されていたら、そのログはサブトランザクション化している可能性があり、必すオペレーションログが続いている（たとえ単一のサブトランザクションログで構成されていても）。

２）オペレーションログ

サブトランザクション化したトランザクション（関数）に与えられた引数をオペレーションログとして保存する。巨大トランザクションを対象としたＢＥＧＩＮマークの直後には必ずオペレーションログを配置する。オペレーションログは、最初にトランザクションへ渡された情報（このファイルをこれだけのサイズトランケートするとか、このファイルをこれだけアペンドライトするなど）である。

【０１７５】リプレイ時には、リプレイヤがファイルシステムを直接操作して、ここで残されたオペレーションログを実行しなければならない。オペレーションログは以下のメンバで構成される。

・パラメタ１

・パラメタ２

・・・

リプレイヤはＢＥＧＩＮマーク内のトランザクションタイプを見ることによって、オペレーションログ部分に並ぶパラメタ群のサイズ及び内容を知ることができる。

３）ヘッダ

一般ログのヘッダと同じ。

４）更新内容

一般ログの更新内容と同じ。

５）５’）ＥＮＤマーク、最終ＥＮＤマーク

一般ログのＥＮＤマークと同じ意味合いを持ち、ＢＥＧＩＮマークに対応するＥＮＤマークあるいは最終ＥＮＤマークが見つからない場合には、ＢＥＧＩＮマーク以降の更新内容をリプレイしてはならない。

【０１７６】巨大トランザクションがサブトランザクションに分割されず、単一のログブロックのみの場合には、ＥＮＤマーク部分には最終ＥＮＤマークが書き出される。複数のサブトランザクションログに分かれている場合には、最後のログブロックだけ、ＥＮＤマークが最終ＥＮＤマークに化ける（最後以外のログブロックには通常のＥＮＤマークが書かれている）。

【０１７７】ＥＮＤマークは一般ログのＥＮＤマークと同一であり固有数値６４バイト、マジックワード、カウンタ（ログ番号）、ログブロックサイズが記録されており、ブロック境界まで固有数値が埋まる。

【０１７８】最終ＥＮＤマークとＥＮＤマークの差は、マジックワードのみである。最終ＥＮＤマークがトランザクション全体の終了を表すことから、とても巨大なトランザクションが多くのサブトランザクションログに分割されている場合には、全ての処理が完了する時にこの最終ＥＮＤマークを出力する。すなわち、巨大トランザクションとして定義されるトランザクションについて、通常のＥＮＤマークで終わるログブロックがいくつか見つかったのに、最終ＥＮＤマークで終わるログブロックが存在しない場合には、それは巨大トランザクションの途中でシステムダウンが発生したことを意味し、ログリプレイヤがオペレーションログのリプレイを行わなければならないことを意味する。

（２）ログの採取

（２．１）ログバッファの管理

ログをトランザクション単位にログボリューム１２０へ出力するためには、メタデータの更新情報を一度、メモリ内に溜める必要がある。これをログバッファと呼ぶ。ログバッファはマウント時にまとめて用意し、ファイルシステム利用中にメモリの追加獲得は行わない。

【０１７９】トランザクション毎のログ採取とし、また、トランザクション動作中に他のトランザクションが並行動作することを狙い、ファイルシステム毎にログバッファを複数持つ。その数は並行動作を許すトランザクション数によって決められる。

【０１８０】並行動作数を限定することはログ機能追加による性能劣化要因の１つとなるが、

・ログリプレイ時の処理を単純化

・巨大なログバッファを分割して使うことによる複雑さの回避

などのメリットが考えられるため、この方式とする。

【０１８１】各トランザクションは開始時に割り振られたログバッファにログを作成し、トランザクション終了時にまとめてログライトバッファへコピーする。ログライトバッファの内容を実際にＩ／Ｏ出力するのはログライトデーモン１０５と呼ぶデーモンの働きによる。

【０１８２】巨大トランザクションには、一般トランザクションが用いるログバッファよりも大きいログバッファ（以降、それぞれを一般ログバッファ、巨大ログバッファと呼ぶ）を１つ用意する。巨大トランザクションも当初は一般ログバッファの１つを用いる。巨大トランザクションとログバッファの関係には次の二段階がある。

１）あまり多くのメタデータを更新せず、一般ログバッファで十分足りると判断できた時点で、次の巨大トランザクションのＢＥＧＩＮを受け付ける。

２）更新するメタデータ数がある程度多く、一般ログバッファでは足りないと判断できた時点で、巨大ログバッファにこれまでの内容をコピー、そこで処理を継続する。

３）更新するメタデータ数が大変多く、巨大ログバッファでは足りない場合には、サブトランザクションとして分割する。

【０１８３】２）におけるバッファ間のコピーが負担となりそうであるが、巨大トランザクションがそれほど多くないと考えると、あまり頻繁に発生するものではないので、この方法とする。

【０１８４】一般ログバッファは状態（利用中・未利

用）について、ビットマップで管理される。各ログバッファを管理する構造をログバッファ数の配列で確保し、それぞれが
・ログバッファのアドレス
・これまでに採取したログサイズ
を持っている。

【０１８５】また、ログバッファの内容は専用のログライトデーモン１０５によってログボリューム１２０へ反映するが、そのログライトデーモン１０５が複数のログバッファの内容を一括してＩ／Ｏ発行するために、トランザクション終了時にそれぞれのログバッファの内容をログライトバッファ１５０と呼ぶ専用の領域にコピーする。

【０１８６】ログライトバッファを管理する構造として、
・追加モードにあるログライトバッファはどちらか
・それぞれのログライトバッファに溜められたログサイズ
がある。

【０１８７】ログライトバッファの内容をログボリューム１２０へ書き出している間は、その後のトランザクションが終了するためにログライトバッファにログバッファをコピーしようとしても、Ｉ／Ｏ中であるためにガードしなければならない。これは性能劣化に繋がるため避けなければならない。そのため、ログライトバッファは２本用意する。

【０１８８】一般トランザクションはトランザクション開始時に、空のログバッファをビットマップより見つけ（未利用ログバッファを０で示し、ここで１とする）、そのビット番号がトランザクション番号となる。

【０１８９】巨大トランザクションが一般ログバッファにて処理進行中、一般ログバッファでは入りきらないと判断されるとこれまでの内容をこの巨大ログバッファへコピーし、そこで処理を継続する。この時、これまで用いていた一般ログバッファは空き状態として、次のトランザクションによる利用を可能にする。

（２．２）巨大トランザクションの管理

空き領域管理ツリーや獲得済領域管理ツリーを操作するトランザクションを巨大トランザクションと定義する。巨大トランザクションは場合によっては木構造を大きく変更しなければならないかもしれない。この時、サブトランザクション化する必要が生ずる。

【０１９０】サブトランザクション化した巨大トランザクションが並行動作した場合、どれもが多くのメタデータを更新すると、メタキャッシュ１３０がいずれ全てダーティとなり、新たにメタデータを読み込んで更新しようにも追い出すこともできずにデッドロックに陥ることが考えられる。

【０１９１】そのため、巨大トランザクションについては並行動作ができないよう、ここではシリアライズした。しかし、上記の通り、多くのメタデータを更新する可能性があると考えられるトランザクションは多い。これらを全てトランザクション終了までシリアライズすると、その性能インパクトは大きい。

【０１９２】巨大トランザクションも、場合によってはそれほど多くのメタデータを更新しないこともありえる。一般トランザクションと同等の数しかメタデータを更新しないのであれば、扱いを一般トランザクションと同様にすることによって、その性能インパクトを小さくすることが可能である。

【０１９３】そこで、巨大トランザクションが一般トランザクションと変わらない程度しかメタデータを更新しないと分かった時点で、この巨大トランザクションの扱いは一般トランザクションと同じにする（デグレード）。

【０１９４】巨大トランザクションはその開始時には一般ログバッファの１つを一般トランザクション開始時と同様に与えられる。しかし、１つの巨大トランザクションが動作中は次の巨大トランザクションにログバッファを与えない点が一般トランザクションの場合と異なる。

【０１９５】巨大トランザクションがある程度以上のメタデータを更新することが分かった時点で、これまでの一般ログバッファの内容を巨大ログバッファへコピーし、そこで処理を継続する。

【０１９６】しかし、それほど多くのメタデータを更新しないと判断されれば、そのまま一般ログバッファで処理が継続され、また、次の巨大トランザクションに対し、処理の開始（ログバッファの使用）を許可する。

【０１９７】こうして、巨大トランザクションを途中から一般トランザクションとして扱うこと（デグレード）を実現する。

（２．２．１）デグレード契機・サブトランザクション化契機

巨大トランザクションは複数のサブトランザクションに分割される場合もあれば、一般トランザクションへデグレードする場合もある。それぞれの判定基準は以下の通りである。

◎デグレード判定基準
・一般ログバッファの残りサイズで、将来全てのメタデータ更新が完了できると判断できた場合。

◎巨大トランザクション用ログバッファへの移行契機
・一般ログバッファの残りサイズが次のステップの最大変更量よりも少ない場合。

◎サブトランザクション判定基準
・巨大ログバッファの残りサイズが次ステップの最大変更量よりも少ない場合。

（２．３）一般トランザクションのログ採取

一般トランザクションは、以下の手順に従いログを採取する。

１）ＬＯＧ＿ＢＥＧＩＮ

ａ）利用するログバッファを確定する。
【０１９８】ｂ）ログバッファの先頭にＢＥＧＩＮマークを作成する。
２）各メタデータの更新
ｃ）更新されたメタデータがリリースされる（ロックが解放される）直前に、更新されたメタデータについて、ヘッダ（更新情報）をログバッファに作成し、更新内容をログバッファにコピーする。
【０１９９】ｄ）更新されたメタデータをリストに繋ぐ。
３）ＬＯＧ＿ＥＮＤ
ｅ）ＥＮＤマークをログバッファに作成する。
【０２００】ｆ）ログバッファをログライトバッファへコピーする。
ｇ）ログライトデーモン１０５がログボリューム１２０へ反映する。
ｈ）このトランザクションで立てられた追い出し不可フラグを落とす。
（２．３．１）ＬＯＧ＿ＢＥＧＩＮ
トランザクション開始を意味する。同一関数内にＬＯＧ＿ＥＮＤの宣言が必要。並行動作可能数だけ既にトランザクションが動作していたら、それらのどれか１つが終了するまで寝て待つ。
【０２０１】ａ）現在実行中のトランザクション数がログバッファの数と等しい場合、それは既に許された並行動作数だけトランザクションが実行中であることを意味する。そのため、他トランザクションが終了するまで寝る。そのような状態でなければ、ログバッファの利用状況を管理するビットマップより未利用状態のログバッファを１つ選び出し、そのビットを立てる。さらに、並行動作数カウンタをインクリメントする。
【０２０２】ｂ）前述のＢＥＧＩＮマークをログバッファの先頭に作成する。
（２．３．２）各メタデータの更新
メタデータを参照した場合にはログ採取の必要はない。更新した場合のみログを採らなければならない。
【０２０３】ｃ）メタキャッシュ１３０に読み込んだメタデータについて処理が終了し、メタキャッシュ１３０より解放するための関数（リリース関数）を呼び出す際、「更新した」ことを明示された場合、ログを採取する。
【０２０４】ｉノードについてはログ対象トランザクション内でのロック解放時がログ採取契機である。具体的には、上記で使用権を得たログバッファにヘッダを作成し、メタデータの更新内容をコピーする。このとき、メタデータの種類によってその手法が若干異なる。
【０２０５】ｃ－１）ビットマップの場合：ヘッダには獲得（解放）したメタデータ番号を入れ、更新内容は「０→１」「１→０」とビット操作だけとする。
ｃ－２）メタデータ本体の場合：更新内容にはメタデータをそのままコピーする。
【０２０６】ｃ－３）スーパブロックの場合：巨大トランザクションがデグレードした場合のみ、一般ログ内でのスーパブロックのログ採取がありうる。その時刻（シーケンシャル番号）とデータ空きサイズを記録する。
【０２０７】コピーするとともに、ログバッファのターミナルで管理されるログサイズに、ここで作成したログのサイズを加える。
ｄ）メタキャッシュ１３０の大きさは有限であることから、トランザクションが進行するためには、必要のないメタデータをメタキャッシュ１３０から追い出し、その場所に必要なメタデータを読み込むという処理を行う追い出し機構がある。更新されたメタデータはログを書き出すまでは追い出されてはならない。そのために、このメタデータをメタライトリストと呼ぶ、メタボリュームへの反映を司るメタライトデーモン１０４が参照するリストへ繋ぐ。
（２．３．３）ＬＯＧ＿ＥＮＤ
トランザクションの終了を示す。ここで、ＥＮＤマークの作成を行う。ログボリューム１２０への反映はログライトデーモン１０５による作業である。
【０２０８】ｅ）ＬＯＧ＿ＢＥＧＩＮにて作成したＢＥＧＩＮマークに対応するＥＮＤマークをログバッファの末尾に作成する。
ｆ）利用したログバッファの内容を追加モードにあるログライトバッファにコピーする。この時にログ番号を決定し、ＢＥＧＩＮマーク、ＥＮＤマークに埋め込む。ログバッファのターミナルに記録していたログのサイズをログライトバッファのサイズに加える。ログバッファの「ターミナル」とは、そのログバッファ自身の構造に関する情報を格納する場所を意味している。同期書き込み要求の場合は、ログライトデーモン１０５（詳細後述）が正常にログボリューム１２０に反映したことを待つ必要があるが、同期書き込み要求でない場合には、ログバッファを解放（ビットを落とす）して終了する。
【０２０９】ｇ）後述するログライトデーモン１０５が、ログライトバッファの内容をログボリュームへ反映する。
ｈ）リリース時に立てた追い出し不可フラグを全て落とす。ただし、フラグを落とすのはログライトデーモンによってなされるため、このトランザクションはそのまま終了する。
（２．４）巨大トランザクションのログ採取
巨大トランザクションも、一般トランザクションとほぼ同様の処理だが、最大の違いは、トランザクションの状況によってログバッファをサイズの大きい巨大ログバッファへ移行したり、サブトランザクションとして分割したログ採取を行わなければならない場合があることである。
１）ＬＯＧ＿ＢＥＧＩＮ

ａ）利用するログバッファを確定する。
【０２１０】ｂ）ログバッファの先頭にＢＥＧＩＮマークを作成する。
ｃ）ＢＥＧＩＮマーク作成と同時に、オペレーションログをログバッファに作成する。
２）各メタデータの更新
メタデータがリリースされた時に「更新したこと」が明示されたら、
ｄ）更新メタデータが空き管理情報の場合、更新メタデータがリリースされる度にＢＴＦリストあるいはＢＴＡリストとよぶ空き管理メタデータのために用意するリストに繋ぐ。この時、既にリストに繋がれていればリスト構造には手を加えない。
【０２１１】ｅ）更新メタデータが空き管理情報以外の場合、更新されたメタデータがリリースされる度にヘッダをログバッファに作成し、更新内容をログバッファ域にコピーする。
【０２１２】ｆ）更新メタデータをリストに繋ぐ。まだ一般ログバッファで処理している場合、
ｇ）一般ログバッファで足りるか判定する。
【０２１３】ｇ－ａ）足りないなら巨大ログバッファへこれまでの内容をコピー。
ｇ－ｂ）足りると確定でき、かつ、デグレード条件を満たせば、次の巨大トランザクションを受け付ける。
【０２１４】ｇ－ｃ）まだ判断ができなければそのまま継続する。既に巨大ログバッファで処理している場合、
ｈ）サブトランザクション化するか判定する。
【０２１５】ｈ－ａ）するなら、ＢＴＦリスト及びＢＴＡリストをたどりながらそれらをログバッファにコピー、ＥＮＤマークを作成し、ログライトバッファへコピーする。ログライトデーモン１０５を起動し、ログを出力する。
【０２１６】ｈ－ｂ）まだしないなら、そのまま継続する。
３）ＬＯＧ＿ＥＮＤ
ｉ）最終ＥＮＤマークをログバッファに作成する。
【０２１７】ｊ）ログバッファをログライトバッファへコピーする。
ｋ）ログライトデーモン１０５がログボリューム１２０へ反映する。
ｌ）このトランザクションで立てられた追い出し不可フラグを落とす。
（２．４．１）ＬＯＧ＿ＢＥＧＩＮ
巨大トランザクションも最初は一般ログバッファを用いる。
【０２１８】ａ）一般トランザクションの場合は、現在の並行動作トランザクション数とログバッファの数との関係によってログバッファを獲得できるかどうかが定まった。巨大トランザクションの場合は、現在他の巨大トランザクションが動作中か否かが問題であり、一般トランザクションの並行動作数とは関係しない。巨大トランザクションが動作中か否かのフラグを調査し、非動作中であれば、そのフラグを立て、ログバッファの利用状況を管理するビットマップより未利用状態のログバッファを１つ選び出し、そのビットを立てる。巨大トランザクションが現在動作中であれば、終了まで寝て待つ。
【０２１９】ｂ）ＢＥＧＩＮマークを作成する。ここで、トランザクションタイプに巨大トランザクションとなりうるトランザクションが指定されていた場合には、必ず後ろにはオペレーションログが付随する。対応するＥＮＤマークがないログはリプレイしてはならない。また、サブトランザクション化しているのに、最終ＥＮＤマークがないなら、オペレーションログのリプレイが必要である。
【０２２０】ｃ）巨大トランザクションをサブトランザクションログとして分割する場合にはオペレーションログを採取する必要がある。ここで、オペレーションログとは、各ログ採取対象関数に与えられたパラメタが、将来リプレイすることが可能な形に変更されたものである。具体的には、メモリアドレスで与えられるパラメタは、メタボリューム内のオフセットに変更して記録する。
（２．４．２）各メタデータの更新
ｄ）更新メタデータが空き管理情報である場合、一回のトランザクションで同一の領域が何度も変更される場合が考えられる。その都度、ログを採取しているとログバッファやログボリュームがフルになる可能性が高まる。これを回避するために、更新情報をリストによって管理し、複数回更新された空き管理メタデータを１つにまとめてログに記録する。具体的には、メタデータの状態毎にリスト管理する専用のＢＴＦリスト及びＢＴＡリストに繋がれる。このリストに繋がれていれば、そのデータはダーティであることを意味する。初めて更新するデータであれば、ターミナルから繋がるリストに追加する。既にリストに繋がれているのであれば、２回目以降の更新であることを意味し、ポインタについてはそのままで良い。リストにメタデータを追加する場合には、ログサイズを更新する。
【０２２１】ｅ）更新メタデータが空き管理情報以外であれば、同一メタデータに対して何度もホールド／リリースが発行されるとは考えられないので、リリースの度（ロック解放の度）にログバッファへコピーする（一般トランザクションの場合と同じ）。ログサイズを更新する。
【０２２２】ｆ）既に何度か述べたが、更新されたメタデータがログよりも先にメタボリューム１１１～１１３へ反映されてはならない。この追い出し不可状態もリスト構造によって管理される。
【０２２３】ｇ）この巨大トランザクションが更新するメタデータが、以降、どれだけの数のメタデータを更新

するかを算出することによって、処理が分かれる。
ｇ－ａ）現在利用している一般ログバッファの残りでは次ステップの実行によるメタデータの更新の全てをまかないきれない場合があると判断した場合には、巨大ログバッファへの移管を行う。
【０２２４】ｇａ－１）これまでに溜まったログバッファの内容を巨大ログバッファへコピーする。この時、ＢＴＦリストはそのまま繋いだままとし、ＢＴＡリストは巨大トランザクション用のターミナルへ移動する。
【０２２５】ｇａ－２）巨大ログバッファを利用して、その巨大トランザクションは処理を継続する。
ｇ－ｂ）現在利用している一般ログバッファの残りだけで、以降のメタデータ更新を全て記録できると判断できるならば、この巨大トランザクションは一般トランザクションにデグレードする。ＢＴＦリストにメタデータが繋がれている場合はデグレード条件を満たさないため、ここでの処理はありえない。ＢＴＦリストはそのまま利用を続ける。
【０２２６】ｇｂ－１）巨大トランザクションが動作中か否かを示すフラグを落とす。
ｇｂ－２）一般トランザクションの並行動作数カウンタをインクリメント。
ｇｂ－３）次の巨大トランザクションが寝ているならば起こして、実行を受け付ける。
【０２２７】ｇｂ－４）このまま一般ログバッファを利用して処理を継続する。
ｇ－ｃ）まだ判断できないのであれば、このまま一般ログバッファを利用して処理を継続する。
【０２２８】ｈ）サブトランザクション判定基準（前述）に基づき、このトランザクションをサブトランザクション化するかどうか判定する。
ｈ－ａ）サブトランザクションに分割する場合は、
ｈａ－１）更新された空き管理情報を、リストを辿りながらログバッファへコピーする。
【０２２９】ｈａ－２）ＥＮＤマークを作成する。
ｈａ－３）ログライトバッファへコピーする。
ｈａ－４）ログライトサイズを更新する。
【０２３０】ｈａ－５）ログライトデーモン１０５を強制起動する。
ｈａ－６）ここで変更した全てのメタデータの追い出し不可フラグを落としてメタライトリストに繋ぐ。これにより，メタライトデーモン１０４は任意の時ににこれらのメタデータを書き出すことが可能となる。
【０２３１】ｈａ－７）巨大ログバッファの先頭にＢＥＧＩＮマークを作成する。
ｈａ－８）オペレーションログを作成する。
ｈ－ｂ）サブトランザクションに分割する必要がない間は、巨大ログバッファ内でそのまま継続する。
（２．４．３）ＬＯＧ＿ＥＮＤ
トランザクションの終了を示す。ここで、最終ＥＮＤマークの作成を行う。実際のログボリューム１２０への反映はログライトデーモン１０５の仕事である。
【０２３２】ｉ）最終ＥＮＤマークを作成する。最終ＥＮＤマークは通常のＥＮＤマークとマジックワードが異なるだけである。
ｊ）利用したログバッファの内容を追加モードにあるログライトバッファへコピーする。この時にログ番号を決定し、ログサイズをログライトサイズに加える。同期書き込み要求の場合は、ログライトデーモン１０５が正常にログボリューム１２０に反映したことを待つ必要があるが、その他の場合には待たないで次の処理へ進む（すなわち非同期書き出しである）。巨大トランザクション動作中フラグを落とし、寝て待つ次の巨大トランザクションを起こす。
【０２３３】ｋ）ログライトデーモン１０５が、ログライトバッファの内容をログボリュームへ反映する。
ｌ）更新したメタデータをログバッファへコピーした時に立てた追い出し不可フラグを全て落とす。ただし、フラグを落とすのはログライトデーモン１０５によってなされるため、このトランザクションはそのまま終了する。
（２．５）ログライトデーモン
並行して動作し、順次終了するトランザクションによって更新されたメタデータのログは専用の書き出しデーモン（ログライトデーモン１０５）によってログボリュームへ反映する。このデーモンはマウント時に起動され、アンマウント時に停止する。すなわち、ファイルシステム毎にスレッドを生成する。
【０２３４】各トランザクションが終了すると、ログバッファ内にＥＮＤマークが作成され、このログバッファの内容はログライトバッファ１５０にコピーされる。このログライトバッファ１５０は複数のログバッファの内容を一括してログボリューム１２０に反映するためのものであり、そこにはメモリコピーの負担があるが、何度もＩ／Ｏを発行するよりは良いと考える。
【０２３５】ログライトバッファ１５０へコピーされると、そのログバッファは利用中バッファリストから空きバッファリストへと繋ぎ直され、かつ、トランザクションが同期書き込み要求でなければ、動作中トランザクション数をデクリメントする。これにより、次トランザクションの実行が進むようになる。
【０２３６】ログライトデーモン１０５はログライトバッファの内容を、定期的、あるいは同期書き込み指定のトランザクションがある場合などにログボリュームに反映する。反映している間（Ｉ／Ｏ中）は２本あるログライトバッファのうち、もう片方のログライトバッファに内容をコピーしてそのトランザクションは処理を終了する。
（２．５．１）処理手順
ログライトデーモン１０５単体の処理手順はそれほど複

雑ではない。
【０２３７】ａ）現在のログライトバッファをライトモードにし、もう片方を追加モードにする。
ｂ）ログライトバッファの内容をログボリューム１２０へ同期書き出しする。
【０２３８】ｏ）場合に応じて有効範囲情報を書き出す。
ｄ）ログ待ちリストに繋がるメタデータを、メタライトリストへ繋ぎかえる。
ｅ）規定時間の間、スリープする。
【０２３９】ｆ）規定時間が経過、または他の要因で起こされたらａ）へ。
以下、詳細説明である。
ａ）Ｉ／Ｏ中はその対象域に対して追加更新は許されない。そのため、現在のログライトバッファが書き出しが終わり、次の書き出しが始まるまでは、もう片方のログライトバッファへ終了したログバッファをコピーするよう設定する。Ｉ／Ｏ中のログライトバッファをライトモード、ログバッファの内容をコピーする方を追加モードと呼ぶ。切り替えはこのデーモンによって行われ、各トランザクションは常に追加モードにあるログライトバッファに自ログバッファの内容をコピーする。
【０２４０】ｂ）ログライトバッファに新しい内容が含まれていないのであれば、Ｉ／Ｏを発行する必要はない。そこで、まずログライトサイズを調べ、ゼロであればＩ／Ｏを発行せず、タイマを指定して再び寝る。書き出すべきログが存在するのであれば、以下の手順に従う。
【０２４１】ｂ－ａ）ログボリュームの空きサイズ判定。ログボリュームの先頭オフセット（Ａ）、メタライトリスト先頭のメタデータを管理する構造の上書き可能オフセット（Ｂ）を調べる。それとログライトオフセット（Ｃ）、及びログボリュームの最終オフセット（Ｄ）から以下の手順となる。
・Ａ＜Ｂ≦Ｃ＜Ｄ
①ログライトサイズが（Ｄ－Ｃ）以下ならば、Ｃから書き出す。
②ログライトサイズが（Ｄ－Ｃ）より大きく、かつ、（Ｂ－Ａ）以下ならば、Ａから書き出す。
・Ａ≦Ｃ＜Ｂ≦Ｄ
③ログライトサイズが（Ｂ－Ｃ）より小さければ、Ｃから書き出す。
【０２４２】これらの条件に合致せず、ログの出力ができない場合には、メタライトデーモン１０４を起動して、自分はスリープする。メタライトデーモン１０４の動作により起動されたら、再度、空きサイズ判定から行う。
【０２４３】ｂ－ｂ）ログボリューム１２０へ同期書き出しする。
ｂ－ｃ）エラー判定。同期書き出し後、エラー判定を行う。エラーが発見された場合の処置については省略する。
【０２４４】ｂ－ｄ）領域判定。
書き出しが終わった後、再び、ｂ－ａ）と同様な空き領域の判定を行う。ここで、残りが少ないと判定された時、その残りの大きさに応じてメタライトデーモン１０４を「平常起動」または「緊急起動」する。
【０２４５】ｃ）ファイルシステム復元に必要なログは、メタライトリスト先頭の上書き可能オフセットからたった今書き出した位置までである。そこで、それを有効範囲情報としてディスクへ書き出す。ここで、書き出しはログのシーケンシャル性を考慮し、ある程度の間隔をおいて行う。ここでは、回数に応じて、数回に一回、書き出すこととした。
【０２４６】ｄ）ｂ）にて書き出したログに含まれるメタデータは全て「ログ未反映状態」、すなわち追い出し不可状態としてリスト（ログ未反映リスト）に繋がれているはずである。そこで、ログ未反映リストを辿りながら、そこに繋がるメタデータをメタライトリストに追加することにより、メタライトデーモン１０４が任意の時にこれらのメタデータをメタボリュームに反映できるようになる。
【０２４７】ｅ）ログもシステム全体から見れば非同期書き出しとし、トランザクションが同期書き込み要求でなければ、ログを書き出す前にそのトランザクションは終了することができる。さらにＩ／Ｏ発行回数を削減するために、複数のトランザクションを一括して書き出すために、しばらくの間スリープし、その間にログライトバッファへ複数トランザクションのログを溜める。
【０２４８】ｆ）規定時間後には自分で起きて、処理を繰り返す。しかし、他要因によって突然起こされて処理を始める場合もある。「その他の要因については次の項（２．５．２）で述べる。
（２．５．２）動作契機
ログライトデーモン１０５は以下の契機でログライトバッファの内容をログボリュームへ書き出す。
◎一定周期
ログライトデーモン１０５は一定周期毎にスリープ状態から自発的に目覚め、ログライトバッファ１５０の内容をログボリューム１２０へ反映する。
◎同期書き込み要求のトランザクション
トランザクションによっては同期書き込み要求で呼ばれる場合がある。このトランザクションについては、ログの書き出しを待たなければ終了してはならない。そのため、ＬＯＧ＿ＥＮＤ呼び出し後に、明示的にＬＯＧ＿ＳＹＮＣを行う。ＬＯＧ＿ＳＹＮＣはログライトデーモン１０５が寝ている場合には起こし、ログライトバッファの内容をその場で書き出させる。
【０２４９】現在ログライトデーモン１０５がＩ／Ｏ中であったら、ログライトデーモン１０５の処理が終わる

のを寝て待つ。
◎ログライトバッファ不足
周期毎にログライトバッファをクリアしても、大きなログバッファが連続して終了すると、その前にログライトバッファがフルに近い状態になることがありえる。この時にはログライトデーモン１０５が起こされ、ログライトバッファ１５０の内容をログボリューム１２０に書き出してクリアする。
【０２５０】具体的には、あるトランザクションが自ログバッファの内容をＬＯＧ＿ＥＮＤによってコピーしようとした時、ログライトバッファ１５０の残りサイズが自ログよりも小さいと判断された時、ログライトデーモン１０５を起こし、追加モードのログライトバッファ１５０を切り替えてもらい、その間は寝て待つ。
【０２５１】現在Ｉ／Ｏ中であり、かつ追加モードのログライトバッファ１５０が足りなくなった時には、そのトランザクションは待ち合わさざるを得ない。
◎メタキャッシュ不足
トランザクション実行中に、メタキャッシュ１３０域のいずれかのメタデータを追い出さなければ必要なメタデータをメタボリューム１１１～１１３から読み込むことができずに処理が進まなくなる場合が考えられる。
【０２５２】そのため、トランザクションが現在キャッシュ上のメタデータを追い出そうとする時、メタキャッシュ１３０上のメタデータが全てダーティであり、かつ、ログ待ちリストに繋がれたメタデータが存在する場合には、ログライトデーモン１０５を起動する。
◎アンマウント時
アンマウント時には必ず書き出さなければならない。そのため、アンマウント処理の延長でログライトデーモン１０５を起こし、書き出しを行う。この時、アンマウントを実行するため、該当ファイルシステムにメタデータを更新するようなトランザクションは動作していないことが保証される。従って、現在のログライトバッファの内容を書き出せば、それで全てである。
【０２５３】ログライトとバッファの内容を書き出した後、ログライトデーモン１０５は終了する。
（３）メタデータ管理
次に、メタデータの管理方式について説明する。メタキャッシュ１３０内のメタデータはｍｅｔａｌｉｓｔ構造体によって管理される。ｍｅｔａｌｉｓｔ構造体には以下のメンバがある。
・メタデータへのポインタ
・ＴＲＡＮＳ（トランザクション）リストポインタ
・メタライトリストｐｒｅｖポインタ
・メタライトリストｎｅｘｔポインタ
・ログ待ちリストｐｒｅｖポインタ
・ログ待ちリストｎｅｘｔポインタ
・ログシーケンス番号
・ログボリューム内オフセット
・トランザクション番号
・状態フラグ
・メタデータタイプ
・ｂｕｆ構造体実体
（３．１）メタデータの状態遷移
ｍｅｔａｌｉｓｔ構造体には次の６種類の状態があり、４種類のリストに繋がれる。それはｍｅｔａｌｉｓｔ構造体のフラグによって示され、それぞれターミナルを別とするリストに繋げられることによって実現する。全てのｍｅｔａｌｉｓｔ構造体はいずれかのリストに繋がっており、例外はない。
Ａ）空き管理構造更新状態
データ空き領域を管理するメタデータがトランザクションによって更新されると、リストに繋ぎ、将来まとめてログバッファにコピーすることは既に述べた。トランザクション終了時に本リストに繋がれたメタデータが直接ログボリュームへ反映される。リストは並行動作数に応じて必要である。このリストをＢＴＦリストと呼ぶ。
【０２５４】この状態にあるメタデータはまだログバッファへはコピーされておらず、同一トランザクションによって再度更新される場合がある。既にリストに繋がれているメタデータであった場合は、リスト状態は変更しない。当然、メタボリュームに反映してはならない。
【０２５５】ＢＴＦリストは単方向環状リストであり、トランザクション途中状態と同じメンバを用いて繋がれている。
Ｂ）利用域管理構造更新状態
実施例ではファイルシステムをエクステント管理している。ｉノードから導かれる、そのファイルが利用しているエクステントを示す、間接エクステントブロックについても、トランザクション内で１つの間接エクステントブロックが何度も更新される場合が考えられる。そこで、空き管理構造の場合と同様に、トランザクション中は独自のリストに繋ぎ、トランザクション終了時にまとめてログバッファへコピーする。リストは並行動作数に応じて必要である。このリストをＢＴＡリストと呼ぶ。
【０２５６】この状態にある間接エクステントブロックはまだログバッファへはコピーされておらず、同一トランザクションによって再度更新される場合がある。既にリストに繋がれている間接エクステントブロックであった場合は、リスト状態は変更しない。当然、メタボリュームに反映してはならない。
【０２５７】ＢＴＡリストは単方向環状リストであり、トランザクション途中状態と同じメンバを用いて繋がれている。
Ｃ）トランザクション途中状態
トランザクション途中を示す。この状態のとき、該当するメタデータを更新したトランザクションは、ある１つのログバッファを専有しており、既にそこに更新後状態がコピーされている。リストは並行動作数に応じて必要

である。このリストをＴＲＡＮＳリストと呼ぶ。

【０２５８】しかし、まだトランザクションが終了していないことからログがログボリューム１２０へ反映されておらず、本メタデータもメタボリュームへの反映はできない。この状態にあるメタデータはファイルのロックによって保護されているため、他トランザクションから参照・更新されることはない。

【０２５９】ＴＲＡＮＳリストのターミナルは配列になっており、その要素番号はトランザクションが用いているログバッファの番号（トランザクション番号）に対応する。

【０２６０】メタライトリストやログ待ちリストに繋がれているメタデータが繋がる場合がある。ＴＲＡＮＳリストは単方向環状リストであり、ＢＴＦリストが用いるメンバを併用する。

【０２６１】Ｄ）ログ待ち状態

トランザクションは既に終了しているが、ログライトバッファの内容はログライトデーモン１０５によってログボリューム１２０へ反映されるため、この時点ではまだログボリューム１２０に反映されていない状態である。（デーモンは同期ライトを行うが、トランザクションの視点から見れば、ログ反映が非同期に行われているように見える。）

トランザクションが終了する際に、Ｃ）の状態にあるＴＲＡＮＳリスト（巨大トランザクションの場合はＢＴＦリストも）をログ待ちリストに繋ぎかえる。このリストはトランザクションが終了する毎に後方へ伸びるリストであり、ログライトバッファと同数の２本存在する。

【０２６２】この状態にあるメタデータは既にトランザクションが終了しており、ログはログライトバッファにコピーされている。トランザクションが終了していることから、他トランザクションによって参照・更新される可能性がある。この時（他トランザクションによって状態が変化した時）、リスト位置は変更せず、状態フラグだけを変更する。

【０２６３】メタライトリストに繋がれたメタデータが、他トランザクションによって再度更新されたために、トランザクション途中状態になり、そのトランザクションが終了したことによって、ログ待ち状態になった場合には、このメタデータはメタライトリストにも繋がれている。

【０２６４】ログライトデーモン１０５によって、ログの反映が進むと、それに応じたメタデータをメタライトリストへ追加していく。この時、既にメタライトリストに繋がれているメタデータは、そのままの位置でいなければならない。

Ｅ）書き出し可能状態

これは、ログバッファのログボリュームへ反映が終了し、自由にメタデータをメタボリュームへ反映することができるようになった状態である。この状態にあるメタデータは他トランザクションによって参照・更新される可能性がある。この時、リスト位置は変更せず、状態フラグだけを変更する。

【０２６５】繋がるリストはメタライトリストであり、１本だけ存在する。メタライトデーモン１０４はこのリストを参照してメタボリュームへの反映を行う。

Ｆ）Ｉ／Ｏ中状態

この状態にあるメタデータは現在非同期ライトによってメタボリュームに対してＩ／Ｏを投げているが、まだ完了していない。Ｉ／Ｏ途中であるため、他トランザクションは操作することができない（参照することはできる）。メタライトリストにそのまま繋がれている。

（３－２）ログボリュームの上書き判定情報

ログボリュームはサイズが有限であるため、サイクリックに利用し、過去のログを上書きしてシステムは動作する。ここで、過去のログを上書きするためには、そのトランザクションが更新したメタデータが全てメタボリュームに反映されていることが必要である。上書き可能領域を管理するために、以下の構造を設け、メタライトデーモン１０４が変更、ログライトデーモン１０５が参照する。

◎ログシーケンス番号

各トランザクションが終了し、ログバッファの内容をログライトバッファへコピーする毎に与えられる、シーケンシャル番号である。ログボリューム内のＢＥＧＩＮマーク、ＥＮＤマークに含まれる番号と同一である。既にライトリストに繋がるメタデータが再度更新される場合にも、この番号は変更されない。

◎ログボリューム内オフセット

ログボリューム内にはトランザクション毎のログが連続して並んでおり、上書きするためには、それぞれのトランザクションが更新した全てのメタデータがメタボリュームへ反映されている必要がある。

【０２６６】同一のログ番号を持つｍｅｔａｌｉｓｔ構造体は、ここにはそのログの先頭オフセットを挿入する。ＬＯＧ＿ＥＮＤによってログバッファの内容がログライトバッファにコピーされる際、そのアドレスとログボリュームの書き出しオフセットから導かれる。トランザクション毎のログボリューム内オフセットがＴＲＡＮＳリストを辿りながら、トランザクション毎のログボリューム内オフセットをそれぞれのｍｅｔａｌｉｓｔ構造体に設定する。

【０２６７】ログライトデーモン１０５は、ログを書き出す際にメタライトリストの先頭に繋がるｍｅｔａｌｉｓｔ構造体を参照し、現在のポインタにログライトバッファに入っているログサイズを足した位置が、このオフセット値を超えないことを確認した後で書き出しを行う。

（３．３）メタボリュームへの反映

トランザクションによって更新されたメタデータは、ト

ランザクションが終了しログがログボリューム１２０に反映されるまでは書き出されてはならない。そのために、メタキャッシュ１３０上の各メタデータはそれぞれの状態に応じてリストに繋がれ、唯一メタボリュームへの反映が許可されているリストはメタライトリストである。

【０２６８】時間のかかるＩ／Ｏ要求をトランザクション内で行うことを避けるために、メタデータのメタボリュームへの反映はトランザクションとは関係ないところで動作するデーモンに委ねる。このデーモンはメタライトリストだけを意識し、ｍｅｔａｌｉｓｔ構造体内の情報から状態に応じてＩ／Ｏ発行するかどうかを決定し、関連付けられたｂｕｆ構造体を用いてメタデータをメタボリュームへ反映する。

【０２６９】デーモンはメタライトリストに繋がれたメタデータを書き出した後、そのメタデータをリストから外し、ｃｌｅａｎであると設定する。

（３．４）メタライトデーモン

メタライトデーモン１０４は、マウント時に起動され、アンマウント時に停止する。すなわち、ファイルシステム毎にスレッドを起動する。

【０２７０】ログ書き出しの終わったトランザクションが更新したメタデータは、それを管理するｍｅｔａｌｉｓｔ構造体が全てメタライトリストに繋がれている。メタライトデーモン１０４はメタライトリストに繋がるメタデータを、ログボリュームの残りが少なくなったと判断された場合などに非同期でメタボリュームに反映する。反映している間は該当するメタデータは更新することができず、Ｉ／Ｏ終了を待ち合わせることになる。

【０２７１】メタライトデーモン１０４はメタライトリストを意識し、繋がるメタデータをメタボリュームへ反映する。これにより、ログボリュームの上書き可能領域が拡大する。

【０２７２】しかし、メタライトリストに繋がれているメタデータの中にも、再度トランザクションによって更新が進み、メタボリュームへの反映が禁じられているものが存在する場合がある。この時、他のメタデータを書き出すとしても、そのメタデータについては非同期ライトの発行を待ちあわせなければならない。このようなメタデータが存在すると、以降のメタデータを全て書き出せたとしても、上書き可能領域を拡大することができない。

【０２７３】メタライトデーモン１０４は他者から起動されて処理を開始する場合と、自発的に起動する場合がある。以下に処理手順を述べるが、システムの状態（ログボリュームの空きやメタキャッシュ１３０の空きなど）に応じて動作が若干異なる。また、ここで述べる処理手順にはデーモン内だけではなく、ドライバから呼ばれる関数での処理も含まれている。

（３．４．１）自発起動処理

メタライトデーモン１０４はタイマによって自発的に起動して、それまでにメタライトシストに繋がれた書き出し可能なメタデータをメタボリュームに反映する。この時、メタライトリストの全てを書き出す訳ではなく、ある一定の数だけ非同期書き出しを行う。普段から少しずつメタデータの書き出しを行っておくことによって、資源不足による起動を避け、Ｉ／Ｏ負荷分散を図る目的がある。

【０２７４】ａ）自発起動時には、メタライトリストに繋がるメタデータ数を調査する。

ｂ）メタライトリストの先頭から、メタデータの状態を調べる。

ｃ）メタデータが書き出し可能状態であれば、その状態をＩ／Ｏ中状態に変更し、非同期ライトを発行する。

【０２７５】ｄ）ｂ＿ｉｏｄｏｎｅの関数がＩ／Ｏの成功を確認する。

ｅ）ｂ＿ｉｏｄｏｎｅの関数をメタライトリストから外す。

ｆ）規定数だけ繰り返す

ｇ）スリープする。

【０２７６】以下、詳細説明である。

ａ）自発起動間隔として定義する間隔毎にメタライトリストに繋がるメタデータ数を確認する。具体的には、メタライトリストのターミナルに含まれる、リンクされたメタデータ数を調べる。このとき、それほど多くのメタデータが繋がれていない場合には書き出しは行わない。

【０２７７】ｂ）メタライトリストには基本的にはログの書き出しが終わった、書き出し可能状態のメタデータが繋がれている。しかし、トランザクションが終了してメタライトリストに繋がれた後で他トランザクションによって更新されると、そのメタデータだけは書き出し不可の状態に戻ってしまう。そのため、メタライトデーモン１０４はメタライトリストに繋がるメタデータの状態を確認しながら書き出し処理を行わなければならない。

【０２７８】ｃ）現在ポイントするメタデータが書き出し可能状態であれば、その状態をＩ／Ｏ状態に書き換え、ｍｅｔａｌｉｓｔ構造体に含まれるｂｕｆ構造体を用いてＩ／Ｏを発行する。Ｉ／Ｏ中状態のメタデータは他トランザクションから更新されることはない。メタデータのディスクライトは非同期ライトで行い、デーモンはＩ／Ｏの結果を待たずに次の処理へ進む。

【０２７９】メタデータが書き出し不可状態であるなら、そのメタデータは諦め、リストの次に繋がるメタデータに進み、状態を調べる。

ｄ）非同期ライトによって発行されたＩ／Ｏの結果を判定するのは、ｂｕｆ構造体のメンバｂ＿ｉｏｄｏｎｅに組み込まれた関数である。この関数にはｂｕｆ構造体が引数に与えられ、それを元にエラー判定処理を呼び出す。ここでエラーを検出した場合には、異常系処理へ進む（省略）。成功していた場合には、該当メタデータの

ｃｌｅａｎ数をインクリメントする。
【０２８０】ｅ）ｂ＿ｉｏｄｏｎｅに組み込まれた関数はメタライトリストの管理も行う。引数に渡されたｂｕｆ構造体の上を見るとそこはｍｅｔａｌｉｓｔ構造体になっており、そのフラグからＩ／Ｏ中フラグを落とし、メタライトリストから外す。また、メタデータをダーティ状態からｃｌｅａｎ状態へ変更する。これは、メタキャッシュ１３０で管理されるメタデータである場合には、それぞれの管理構造体で管理されており、この管理構造体はｍｅｔａｌｉｓｔ構造体からポイントされる。
【０２８１】全ての処理が終了したら、このｍｅｔａｌｉｓｔ構造体をリリースする。
ｆ）メタライトリストに繋がる書き出し可能メタデータを、次々とリストを辿りながら定義した数だけ処理を繰り返す。ここで、書き出しが不可とされているメタデータについては、この数には含めない。また、メタライトリストが定義数よりも少ない場合には、当然そこで終了となる。
ｇ）スリープする。再度、自発的に起動するか、あるいは資源不足によって他から起動されるまで、スリープは継続する。
（３．４．２）平常処理
システム状態が以下の場合には、ここで述べる処理手順に従いメタライトデーモン１０４は動作する。
◎ログボリュームの残りが少ない。
【０２８２】ログライトデーモン１０５が判断する。ログライトバッファを書き出す際に、上書き可能域の大きさを計算し、これが所定の閾値を下回った時にメタライトデーモン１０４を起動する。
◎メタキャッシュの残りが少ない。
【０２８３】更新リリースがあった時、各メタデータのｃｌｅａｎ数がデクリメントされるが、その数が所定の閾値を下回った時にメタライトデーモン１０４を起動する。
ａ）メタライトリストの先頭から、メタデータの状態を調べる。
【０２８４】ｂ）メタデータが書き出し可能状態であれば、その状態をＩ／Ｏ中状態に変更し、非同期ライトを発行する。
ｃ）ｂ＿ｉｏｄｏｎｅの関数がＩ／Ｏの成功を確認する。
【０２８５】ｄ）ｂ＿ｉｏｄｏｎｅの関数をメタライトリストから外す。
ｅ）メタライトリストの最後まで繰り返す。
ｆ）スリープする。
【０２８６】以下、詳細説明である。
ａ）～ｄ）自発的に起動した場合と同じである。
ｅ）メタライトリストを辿りながら、繋がる全ての書き出し可能メタデータについて処理を繰り返す。平常処理時には、書き出し不可のメタデータについては飛ばして処理を行う。そのため、ログボリューム不足時には、メタライトリストの先頭、すなわち、ログボリュームの上書き可能位置を制限しているメタデータが書き出せなければ資源不足は解消しないが、平常処理であるため、ここでは特別な対処を行わない。
【０２８７】ｆ）タイマを設定してスリープする。
（３．４．３）緊急処理
システム状態が以下の場合には、ここで述べる処理手順に従いメタライトデーモン１０４は動作する。
◎ログボリュームの残りが非常に少ない。
◎メタキャッシュ１３０の残りが非常に少ない。
【０２８８】ａ）トランザクションの新規開始を制限する。
ｂ）メタライトリストの先頭から、メタデータの状態を調べる。
ｃ）メタデータが書き出し可能状態であれば、その状態をＩ／Ｏ中状態に変更し、非同期ライトを発行する。
【０２８９】ｄ）ログ未反映状態となっているメタデータがあれば、ログライトデーモン１０５を起動する。
ｅ）ｂ＿ｉｏｄｏｎｅの関数がＩ／Ｏの成功を確認する。
【０２９０】ｆ）ｂ＿ｉｏｄｏｎｅの関数をメタライトリストから外す。
ｇ）繰り返す。
ｈ）現在の資源状態を調査し、不足があれば、それが解消するまで繰り返す。
【０２９１】ｉ）トランザクションの受付を開始する。
ｊ）スリープする。
以下、詳細説明である。
【０２９２】ａ）緊急時には、新規のトランザクションの開始を受け付けない。具体的には、トランザクションに利用するログバッファを与えないことによって、そのトランザクションのＢＥＧＩＮ宣言時にてスリープさせる。そのために、特定のフラグを設け、ＢＥＧＩＮ宣言時にそのフラグを参照しなければならない。
【０２９３】ｂ）～ｇ）ここは基本的に平常処理の場合と同じ処理である。すなわち、メタライトリストの先頭から、追い出し不可状態のメタデータは飛ばして、書き出し可能状態のメタデータを順に非同期ライトによって書き出す。
【０２９４】ただし、ｄ）だけ異なる。
ｄ）ログボリュームへの書き出しがデーモンによって行われることから、たとえトランザクションが終了していてもログが未反映のためにメタボリュームへの反映を拒否しているメタデータが存在することが考えられる。この場合は、強制的にログライトデーモン１０５を起動して、書き出し可能状態へ移行するように操作する。既にログライトデーモン１０５が動作中であれば、そのまま先へ進む。
【０２９５】ｈ）ここで、現在の資源状態を調べる。平

常処理の動作契機となる閾値以上の資源が回復していなければ、再度、メタライトリスト内のメタデータ書き出しを試みる。ここで、ｄ）によってログ未反映状態のメタデータが、書き出し可能となったことによって進展することを期待している。複数回、繰り返すことによって、新規トランザクションの受付を制限していることから、資源回復までメタデータの書き出しが可能であると考える。

【０２９６】ｉ）ａ）にて制限していたトランザクションの受付を再開する。

ｊ）タイマを設定して、スリープする。

（４）獲得解放処理

メタデータの割り当て状況を管理するビットマップの状態として、FREE-dirtyとALLOC-dirtyを区分し、FREE-dirtyなビットマップからは獲得しないようにする。

【０２９７】具体的には以下の通りである。

・マウント時にビットマップを読み込む際に、１つを獲得用と定める。説明を簡単にするため、複数読み込むビットマップのうち、一番若いもの（最初に読み込むもの）を最初の獲得用ビットマップとする。

・獲得用ビットマップの複製を作成する。

・獲得時には複製を検索して獲得位置を決定し、本体と複製の両方のビットを操作する。

・解放時には本体のみのビットを操作する。

・複製ビットマップには解放が記録されないため、獲得処理が進むと全てのビットが立ち、そのビットマップでは獲得できないと判断される。その場合、現在メモリ上にあるビットマップのうち、ＣＬＥＡＮなもの、またはALLOC-dirtyのビットマップを次の獲得用ビットマップと定義し、その複製を作成する。

・メモリ上のビットマップが全てFREE-dirtyである場合には、どれかを追い出し、新しいビットマップを獲得用ビットマップとして読み込む。そして、その複製を作成する。ここで、追い出し・読み込みのアルゴリズムは従来通りで良い。

【０２９８】獲得用ビットマップとして選ばれたビットマップは追い出しの対象とはしない。そのためメタキャッシュ域不足によって他から追い出されることはない。また、メタライトリストに繋がったことによって、メタボリュームに反映された場合にも、ＣＬＥＡＮな状態にはなるが、複製は更新せず、そのままとする。

【０２９９】本実施の形態による効果は、以下の通りである。以上説明したように、本発明によれば複数の二次記憶装置に保存されたメタデータのログにボリューム情報を含め、また、有効なログの位置を算出・保存することでリプレイするログ量を減少せしめ、さらに、ログボリューム全体のゼロクリアをする必要をなくすことにより、ログ機構の最大の利点であるところのファイルシステム復元時間の短縮に及ぼす影響を小さくし、オーバオールコンピュータシステム可用性の増大に寄与するところが大きい。

【０３００】さらに、本発明によれば一回トランザクション内で複数回更新されるメタデータのログを一度しか採取せず、また、ログバッファの分割や、獲得・解放処理の特殊なログ採取方式によって複数のトランザクションの独立性を考慮し、かつ、ログ機構導入による速度性能の劣化を最小に留めることにおいて寄与するところが大きい。

【０３０１】加えて、本発明によればトランザクション毎に異なるログの採取量を考慮し、多くのメタデータを更新するトランザクションについては分割し、トランザクションに与えられたパラメタをもログに採取し、リプレイ時にそのパラメタを元に、中途で終わったトランザクションを再度実行することによって、オペレーションのセマンティクスを保証した復元が可能となる面において寄与するところが大きい。

【０３０２】なお、上記の処理機能は、コンピュータによって実現することができる。その場合、説明した処理内容は、コンピュータで読み取り可能な記録媒体に記録されたプログラムに記述されており、このプログラムをコンピュータで実行することにより、上記処理がコンピュータで実現される。コンピュータで読み取り可能な記録媒体としては、磁気記録装置や半導体メモリ等がある。市場へ流通させる場合には、ＣＤ－ＲＯＭ(Compact Disk Read Only Memory)やフロッピーディスク等の可搬型記録媒体にプログラムを格納して流通させたり、ネットワークを介して接続されたコンピュータの記憶装置に格納しておき、ネットワークを通じて他のコンピュータに転送することもできる。コンピュータで実行する際には、コンピュータ内のハードディスク装置等にプログラムを格納しておき、メインメモリにロードして実行する。

【０３０３】

【発明の効果】以上説明したように、第１の発明では、メタキャッシュ内のメタデータとともにメタデータがどのメタボリュームから取り出されたのかを示すメタデータ管理情報をログとして採取するようにしたため、保持されたログがどのメタボリュームのメタデータに関するログであるのかを管理することができる。その結果、複数のログボリュームにメタデータが格納されていても、ファイルシステムの不整合を修正することが可能となる。

【０３０４】また、第２の発明では、ログデータの有効範囲を管理するようにしたため、ファイルシステムを復元する際には、ログボリューム１２内の有効なログのみを用いて効率よく復元処理を行うことが可能となる。

【０３０５】また、第３の発明では、ログデータに対して付与するシーケンス番号の最大値を、システムの使用可能年数以上使い続けることができる値としたことで、常に昇順の採番が可能となり、ログボリュームのゼロク

リアに伴う処理の遅延を避けることができる。

【０３０６】また、第４の発明では、メタデータが複数回更新される場合には、最終形態のみをログとして採取するようにしたため、ログデータが短縮されるとともに、ログデータの短縮に伴いファイルシステム復元時間の短縮が図れる。

【０３０７】また、第５の発明では、割り当て管理情報の一部の複製を獲得操作用管理情報とし、メタデータの獲得時には獲得操作用管理情報内から取得すべきメタデータを特定するが、解放時には獲得操作用管理情報の情報を更新しないようにしたため、メタデータの解放直後に別のトランザクションにより獲得されることがなくなる。その結果、システムダウン時に中途までしか終了していなかった解放トランザクションが解放したはずである領域は、解放される直前の状態のまま保全されることが保証される。

【０３０８】また、第６の発明では、獲得、解放操作のログとして、その操作対象となった情報のみを記録するようにしたため、ログ採取量が少量ですむ。また、第７の発明では、複数のログバッファを設け、さらにいくつか異なるサイズのログバッファを用意し、トランザクション毎のログをそのトランザクションに適したサイズのログバッファに格納するようにしたため、複数のトランザクションが並列実行される際のトランザクションの独立性を高めることができ、また、メモリ空間を有効に活用できる。

【０３０９】また、第８の発明では、１つのトランザクションのログがログバッファに収まらない場合に、中間ログとして完結されたログをログボリュームに書き出すようにしたため、部分的であるログを、ファイル復元時には１つのトランザクションのログと同値に扱うことが可能となり、ファイルシステムの復元時に望ましい状態に復元するのが容易となる。

【０３１０】また、第９の発明では、ログ採取に関するシステムの状態よってトランザクションの受け入れを制限するようにしたため、複数のトランザクションが並列実行されることによるメモリ枯渇等の障害の発生を防止することができる。

【図面の簡単な説明】

【図１】第１の発明の原理構成図である。

【図２】第２の発明の原理構成図である。

【図３】第３の発明の原理構成図である。

【図４】第４の発明の原理構成図である。

【図５】第５の発明の原理構成図である。

【図６】第６の発明の原理構成図である。

【図７】第７の発明の原理構成図である。

【図８】第８の発明の原理構成図である。

【図９】第９の発明の原理構成図である。

【図１０】本発明を適用するデータ処理装置のハードウェア構成図である。

【図１１】ファイルシステム上で動作するログ採取機能の構成図である。

【図１２】メタデータ管理情報を示す図である。

【図１３】ログバッファの形式を示す図である。

【図１４】有効範囲を説明する図である。

【図１５】ログ採取処理のフローチャートである。

【図１６】メタボリュームの割り当て管理状況を示す図である。

【図１７】ビットマップによるメタデータ獲得処理を示すフローチャートである。

【図１８】解放処理のフローチャートである。

【図１９】トランザクションの処理の開始と終了の状況を示す図である。

【図２０】ログバッファへのログの格納状況を示す図である。

【図２１】ログ採取手順を示すフローチャートである。

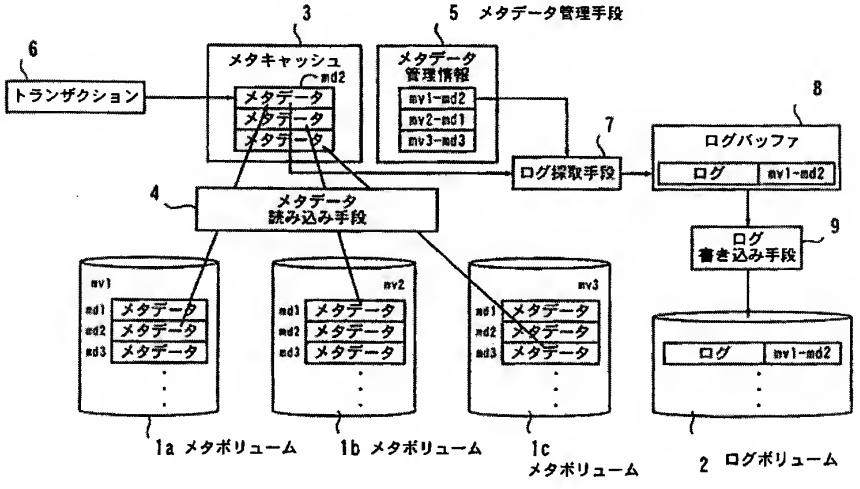
【図２２】ファイルシステム復元処理を示すフローチャートである。

【図２３】新規トランザクションの受け入れ許否判定処理を示すフローチャートである。

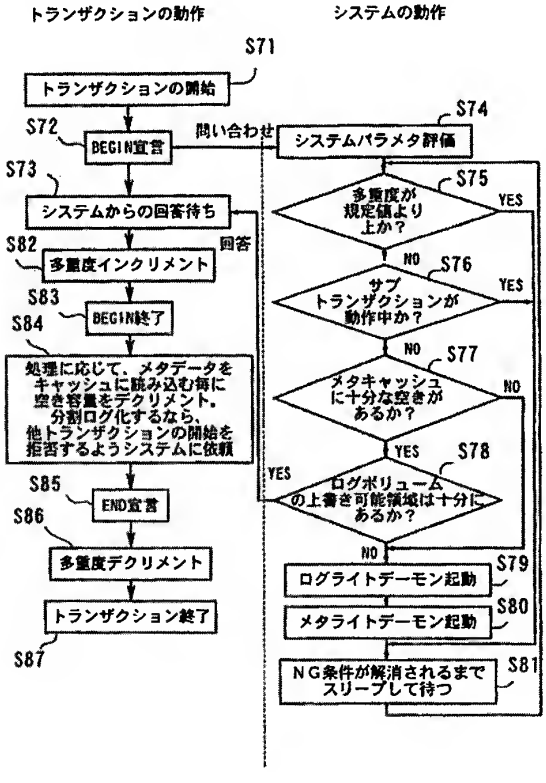
【符号の説明】

１ａ～１ｃ　メタボリューム
２　ログボリューム
３　メタキャッシュ
４　メタデータ読み込み手段
５　メタデータ管理情報
６　トランザクション
７　ログ採取手段
８　ログバッファ
９　ログ書き込み手段

【手続補正２】
【補正対象書類名】図面
【補正対象項目名】図１
【補正方法】変更
【補正内容】
【図１】



【手続補正３】
【補正対象書類名】図面
【補正対象項目名】図２３
【補正方法】変更
【補正内容】
【図２３】



フロントページの続き

(72)発明者　鷲見　昌司
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(72)発明者　山口　悟
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(72)発明者　谷脇　貞善
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(72)発明者　浜中　征志郎
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

Ｆターム(参考)　5B082 CA17 DC06 DC12 DD04 EA02
FA02 FA12 GB05 GB06